2013

ADEPT Runtime/Scalability Predictor in support of Adaptive Scheduling

Gholamhossein Deshmeh
Universty of Windsor

ADEPT Runtime/Scalability Predictor in support of Adaptive Scheduling

By

Gholamhossein Deshmeh

A Dissertation

Submitted to the Faculty of Graduate Studies

through **the School of Computer Science**

in Partial Fulfillment of the Requirements for

the Degree of **Doctor of Philosophy**

at the University of Windsor

Windsor, Ontario, Canada

2013

ADEPT Runtime/Scalability Predictor in support of Adaptive Scheduling

By
Gholamhossein Deshmeh

APPROVED BY:

____
Shikharesh Majumdar, External Examiner,
Dept. of Systems and Computer Engineering,
Carleton University

____
Majid Ahmadi,
Dept. of Electrical and Computer Engineering

____
Joan Morrissey,
School of Computer Science

____
Alioune Ngom,
School of Computer Science

____
Robert D. Kent, Advisor
School of Computer Science

September 4th, 2013

Declaration of Co-Authorship / Previous Publication

I. Co-Authorship Declaration

I hereby declare that this thesis incorporates material that is result of joint research, as follows:

This thesis incorporates the outcome of a joint research undertaken in collaboration with Jacob Machina under the supervision of Dr. Angela Sodan. The collaboration is covered in Chapter 3 of the thesis. In all cases, the key ideas, primary contributions, experimental designs, data analysis and interpretation, were performed by the author, and the contribution of co-authors was primarily through the provision of a method for fast curve fitting and its implementation, the validation of the proposed R-metric, re-factoring the ADEPT code to improve its speed and fix several bugs, and experiments setup. Wai Ling Yee provided hints in regards to the closed-form solution to the Downey model.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from each of the co-author(s) to include the above material(s) in my thesis.

I certify that, with the above qualification, this thesis, and the research to which it refers, is the product of my own work.

II. Declaration of Previous Publication

This thesis includes 1 original paper that has been previously published, as follows:

Thesis Chapter	Publication title/full citation	Publication status
Chapter 3	[A.Deshmeh 2010] Deshmeh, A. Machina, J. Sodan, A., ADEPT scalability predictor in support of adaptive resource allocation, In 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS), April 2010, pp. 1-12.	*Published*

I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as graduate student at the University of Windsor.

I declare that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

A job scheduler determines the order and duration of the allocation of resources, e.g. CPU, to the tasks waiting to run on a computer. Round-Robin and First-Come-First-Serve are examples of algorithms for making such resource allocation decisions. Parallel job schedulers make resource allocation decisions for applications that need multiple CPU cores, on computers consisting of many CPU cores connected by different interconnects. An adaptive parallel scheduler is a parallel scheduler that is capable of adjusting its resource allocation decisions based on the current resource usage and demand. Adaptive parallel schedulers that decide the numbers of CPU cores to allocate to a parallel job provide more flexibility and potentially improve performance significantly for both local and grid job scheduling compared to non-adaptive schedulers. A major reason why adaptive schedulers are not yet used practically is due to lack of knowledge of the scalability curves of the applications, and high cost of existing white-box approaches for scalability prediction. We show that a runtime and scalability prediction tool can be developed with 3 requirements: accuracy comparable to white-box methods, applicability, and robustness. Applicability depends only on knowledge feasible to gain in a production environment. Robustness addresses anomalous behaviour and unreliable predictions. We present ADEPT, a speedup and runtime prediction tool that satisfies all criteria for both single problem size and across different problem sizes of a parallel application. ADEPT is also capable of handling anomalies and judging reliability of its predictions. We demonstrate these using experiments with MPI and OpenMP implementations of NAS benchmarks and seven real applications.

DEDICATION

I dedicate this dissertation to my wife, Golriz Mirarab, who was with me every step of this journey.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my supervisor, Dr. Robert Kent, for his outstanding and continuous support of my PhD studies, which made this dissertation possible. His guidance and deep insights were invaluable to my studies. I also remain indebted to him for his understanding and support during all the difficult times.

I would also like to thank all my committee members: Dr. Majid Ahmadi, Dr. Alioune Ngom, Dr. Joan Morrissey, and the external examiner Dr. Shikharesh Majumdar. Thank you for your great advice.

I would like to thank SHARCNET for partly funding this research. Input data for the experiments was collected on CFI-funded SHARCNET resources. We thank John Morton from SHARCNET for providing us with data on real applications.

I would like to thank my parents, Lotfali Deshmeh and Parvaneh Hashemi for their support.

I would also like to especially thank my aunt Manijeh Deshmeh for all her love and support at each step.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

ADEPT: Automatic Downey-based Envelope-constrained Prediction Tool

Adaptive Resource Allocation: Allocation of resources to different applications based on the current system load, which allows a job scheduler to adapt its decisions to the current status of the system.

Black-box prediction: a category of prediction methods that depend only on external observation of application's behavior, e.g. its runtime.

White-Box prediction methods: a category of prediction methods that depend on internal knowledge of the target application, e.g. the number of iterations in the main loop.

Gray-box prediction methods: a category of prediction methods that combine black-box and white-box observations.

CHAPTER 1

Introduction

A job scheduler determines the order and duration of the allocation of resources, e.g. CPU, to the tasks waiting to run on a computer. Round-Robin and First-Come-First-Serve are examples of algorithms for making such decisions. Parallel job schedulers need to make such resource allocation decisions for applications that need multiple CPU cores, on computers consisting of many CPU cores connected by different interconnects. An adaptive parallel scheduler is a scheduler capable of adjusting its resource allocation decisions based on the current resource usage and demand. This resource allocation method is referred to as adaptive resource allocation. Adaptive parallel schedulers that decide the numbers of CPU cores to allocate to a parallel job provides more flexibility and potentially improve performance significantly for both local job and grid job scheduling compared to non-adaptive schedulers. Adaptive CPU resource allocation is a widely researched topic in job and grid scheduling with potential to improve response times significantly (up to 70%) by reducing fragmentation and considering the current machine load [V.K.Naik 1997][W. Cirne 2003] [A.C.Sodan 2006][L.Barsanti 2006]. Taking the current machine load into account contributes most to the improvement of response times. These improvements are achieved by running applications with more resources if the current machine load is light, and with fewer resources if the load is heavy [V.K.Naik 1997] [A.C.Sodan 2009]. This is due to the typical shape of efficiency curves which describe how well the processor cores allocated to a parallel application are utilized in terms of serial runtime divided by allocated numbers of cores and the corresponding runtime, i.e. diminishing efficiency beyond an application-specific numbers of cores.

Adaptive resource allocation is a practically promising approach, considering that a study found that 98% of the users said their applications could adjust to different resource allocation at start-time [W.Cirne 2003]. Adaptive resource allocation depends on efficiency curves per problem size (strong scaling) since efficiency-based allocation was found superior to uninformed approaches like equal resource partitioning [S.H.Chiang 1996]. However, scalability and efficiency curves, which show the the obtained speedup (serial runtime divided by runtime for a specific number of cores) and utilization of cores for different numbers of allocated cores, are not generally available; this is a major reason why adaptive resource allocation is not yet incorporated in practical schedulers. Thus, providing scalability prediction in an easy-to-use manner would open new possibilities for better practical scheduling. Users may also select job sizes “tactically” under considerations of trading shorter waiting times for increased runtimes.

Scalability prediction is also relevant for determining the maximum meaningful CPU resource allocation to a parallel job (and therefore an often-tackled problem, e.g. [X.H.Sun 1999]) as feedback to users and system administrators. Though so far applied mostly on clusters, with the emergence of parallel computing in every-day life on multi-core systems, adaptive schedulers will likely increase in practical relevance. This is especially true if the resources allocated to a virtual-machine running parallel jobs can vary [A.C.Sodan 2009]. Fortunately, OpenMP applications on multi-core SMP servers were found to exhibit similar shapes of speedup/runtime curves as MPI applications on clusters [M.Curtis-Maury 2005]. This opens the possibility of applying the same scalability prediction approach.

Accurate predictions can be obtained via either black-box or white-box approaches. The latter are based on application-internal and machine information, require code instrumentation, compiler/OS support, analysis of memory-access behavior, simulation, etc. [L.Carrington 2003][B.Lafreniere 2005][G.Marin 2004] [X.H.Sun 1999]. Thus, white-box approaches are complex and computationally expensive, making them unsuitable for large-scale use in supercomputing centers though indispensable for cross-site prediction or projection of performance on not yet practically available platforms. Black-box approaches predict scalability (speedup and runtime) using only runtime observations on different numbers of nodes, by assuming conformity to a simple descriptive model which can be fitted to the observations to derive a specific model instance. The required observations can easily be obtained from data routinely collected in historical databases by supercomputer centers or from explicit test series. This makes black-box approaches much easier and much cheaper to apply. However, to be practical, the number of required observations needs to be small.

We have performed a survey on the existing methods for performance prediction, the result of which is a taxonomy of these methods, as well as details on their strengths, weaknesses, and an analysis of open problems. This survey forms a key contribution of our work, as well as a basis on which we build our hypothesis.

Based on our survey and taxonomy of performance prediction methods, our overall goal is scalability prediction (in the sense of strong scaling), on both multi-core SMP servers and clusters, which is practically feasible for production environments. To enable production use, we apply a black-box approach based on the Downey model shown to capture simplified behavior of parallel applications very well [A.Downey 1997 Model]. The Downey model has been around for a long time but has not been widely used due to many real applications not fully conforming to the model, e.g. by showing super linear speedups, and due to reliability of a specific prediction being hard to judge.

As described in [A.Deshmeh 2010], with the development of ADEPT (Automatic Downey-based Envelope-constrained Prediction Tool), we pursued the following detailed goals:

- Achieve high prediction accuracy, while requiring only few observations (typically 3 to 4).

- Provide a computationally efficient approach for deriving the model instance.

- Identify cases where the application does not fully conform to the Downey model as anomalies, with automatic correction and multi-phase modeling for individual irregular points and typical patterns.

- Perform reliability judgment which recognizes unsuitable observation layout and proposes placement ranges of additional observations.

We decomposed the problem of performance prediction in a production environment into the sub-problems as outlined according to the above requirements. First, we developed a black-box performance prediction tool capable to fitting Downey model instances to observations assumed to conform closely to the model. This provided the basic functionality in ADEPT. We next addressed the challenge of anomalous behavior in parallel applications, by studying different and typical scalability patterns. This resulted in development of a metric for measuring how well-behaved a particular parallel application is, by calculating a magnitude of deviation from the expected behavior. The developed metric was extended to cover applications for which serial runtime is not known. We then studied reliability problems when making performance predictions, and compiled a list of reliability problems and their symptoms. This allowed us to develop responses to each of these challenges for our prediction tool. As result of these steps, ADEPT employs a special envelope-derivation technique which constrains the search for the best-fitting model instance, a special metric for detection of anomalies, and special pattern handling for cases like super-linear speedup. To validate our prediction tool, we studied the evaluation methods used in the literature for performance prediction methods, the results of which are presented in Chapter 2. The result of this study was the selection of one of the most widely used target benchmark set, as well as several real world applications to further ensure applicability of our prediction tool.

Experiments with the NAS benchmarks [D.H.Bailey 1995] and seven real applications show the efficiency and prediction quality of ADEPT in handling normal cases and anomalies. We obtained generally above 80% prediction accuracy, even in cases with anomalies and for predictions which extrapolate for more than twice the number of nodes that were used in the closest observation. The experiments also demonstrate the effectiveness of reliability judgment.

Having achieved highly accurate predictions for a single problem size, we next focus on the performance prediction across problem sizes for a parallel application. The main motivations for this move are: 1) there are potentially significant benefits for a scheduler if such predictions are available; it makes possible adaptive scheduling as users move to larger problem sizes of the same application, and 2) there are not any existing black-box prediction tools that address this issue. We propose an extension to ADEPT which makes it capable of addressing cross problem size performance prediction with the addition of one extra input: the problem size for which the observations are made.

To summarize, the contributions of this dissertation are as follows:

- An extensive survey on the state of the art of performance prediction methods
- A novel performance prediction method, which can be utilized by users and parallel application schedulers to obtain runtime and scalability curves of parallel applications. For schedulers, this can result in significant improvement of performance metrics, as previously described.
- The proposed prediction method is highly applicable, in terms of its requirements, i.e. 3 or 4 observations of runtime on different numbers of cores, and its computational complexity. This makes it feasible for an

adaptive parallel application scheduler to obtain predictions of parallel applications' runtime and scalability despite the constraints of a production environment and the need for predictions on many parallel applications. As described by our survey in the next chapter, this possibility is not offered by other prediction methods.

- The high accuracy of the prediction method, which is comparable to expensive, white-box performance prediction methods.

- The capability of the prediction method to make predictions without assistance from the user or OS-level support

- The capability of the prediction method to handle anomalous behavior by parallel applications, which makes the method robust, further increasing its applicability in a production environment.

- The capability of the presented prediction method to identify unreliable predictions, correct them when possible and generate warnings otherwise, in order to avoid misleading the user of the tool (whether the user is a human or an adaptive parallel application scheduler).

- The capability of the method to make predictions across different problem sizes of a parallel application, thus increasing its applicability.

The rest of this dissertation is organized as follows. In chapter 2, we present the background on performance prediction, as well as our survey which provides a taxonomy of the existing performance prediction methods. In chapter 3 we describe the structure of ADEPT, its contributions, and the experimental results. Chapter 4 draws conclusions and outlines directions for future work.

CHAPTER 2

Background and Literature Review and Analysis

2.1. *Performance Prediction*

Performance prediction is the task of providing an estimation of the performance of an instance of an application on a specific platform, where the application of interest may be serial, parallel, or distributed, and an instance of the application of interest is identified as the combination of input parameters that determine the problem that is being solved as well as the properties of the solution. The platform may be a single CPU, a multi-core desktop machine, a cluster with tens to thousands of cores connected by interconnects, or a distributed grid environment.

Backfilling schedulers, a common type of production scheduler for local scheduling on clusters, depend on performance prediction in terms of jobs' runtime estimations to perform backfilling. Usually the user is asked to provide an estimate of the runtime of the job he/she is submitting, and underestimation is punished by killing the job once it runs past the estimation. Studies have shown that user runtime estimate are generally inaccurate [A.W.Mu'alem 2001]. There have also been several papers in the literature claiming, counter-intuitively, that inaccuracy in runtime estimation actually improves the performance of the scheduler, suggesting better performance of the scheduler if the runtime estimates are doubled [A.W.Mu'alem 2001], [D.Zotkin 1999], or even for randomized runtime estimates [D.Perkovic 2001]. However, these claims were negated by more recent research work. In a keynote speech, [D.Tsafrir, 2010] emphasizes that, despite some previous claims, inaccuracy in runtime estimation does not lead to better scheduler performance. In [D.Tsafrir 2007], authors demonstrate that doubling the runtime estimation improves the performance of the backfilling scheduler, but does so to an even higher degree if the original estimate is accurate, thereby reestablishing the need and motivation for accurate performance predictions.

[S.H. Chiang 2002] emphasizes the importance of accurate runtime prediction by evaluating the performance of a backfilling scheduler on heavy loads and a leading edge production platform, and concluding that accurate estimations can improve the performance of the scheduler much more significantly than was assumed before. Moreover, authors conclude that users who provide accurate runtime estimations will observe performance improvements even if other users do not provide accurate estimates. The improvements are so large that authors suggest the use of test runs to obtain accurate estimations.

[D. Tsafrir 2007] paves the way even further for incorporating performance prediction into production schedulers by separating its two historical roles of providing backfilling information and providing killing times for jobs, i.e. the system does not kill jobs if they are longer than the system generated predictions. Instead, predictions are corrected adaptively if they are proved wrong. [D.Talby, 2006] describe another important application of performance prediction in job schedulers, which is assisting in scheduling of moldable jobs: the scheduler must decide whether it is best to wait and start the job later with more processors, or start the job immediately. This decision has to be based on prior knowledge of jobs' runtimes. [D.Talby, 2006] also proposes a standard interface for all predictors, to increase the applicability to production schedulers. A similar application is proposed by [W.Smith 1999], which uses runtime predictions to estimate queue wait times.

A detailed discussion on the role of performance prediction in various aspects of high-performance computing is presented in [K.J.Barker 2009]. These roles include the design of new machines which uses performance prediction to explore the extremely large design space, the decision of which new platforms to acquire which uses performance prediction to do a cost-benefit analysis, and the installation of new systems which uses performance prediction to verify the installation. [J.Zhai 2010] emphasizes on the role of performance prediction in the studies for acquisition of new systems and proposes a method that accordingly assumes the availability of a single node of the new platform for performance prediction. [D.J.Kerbyson 2002] uses performance modeling and prediction for exploring platform architectures. Similarly, [E.Ipek 2006] discuss the use of performance prediction for making design decisions for new parallel systems. [L.T.Yang 2005] suggests that scientists can choose a parallel system for their application based on prediction of application's performance on available platforms. [K.Davis 2009] accurately predicts the performance of two petascale applications on an HPC platform before and after an upgrade, emphasizing a potential key role for performance prediction in HPC platform upgrade decisions. Performance prediction has also been used for performance tuning of parallel applications [A.Tiwari 2009], [K.Singh, 2010], and for performance tuning and identifying performance bottlenecks [G.Marin, 2007]. According to [R.Sarikaya 2010] performance prediction can be used for the improvement of power-performance decisions in dynamic power management.

A case for the importance of scalability prediction is made in [W.Cirne 2003], by specifying that 98% of the users think their jobs can adapt to different numbers of processors at start-time. A speedup model can assist scientists in deciding whether to make a request for the allocation of a larger numbers of cores on a cluster, e.g. SHARCNET [SHARCNET] holds regular rounds of applications for large numbers of cores on its clusters. Considering the costs associated with making and processing such applications makes a case for a speedup model. [A.Duran 2008] uses speedup prediction to dynamically determine the number of OpenMP threads to create for an application. [Z.Wang 2009] predicts scalability on multicore machines for OpenMP programs. [K.Singh 2010] proposes a method for dynamic concurrency throttling, which is reducing the number of threads of an application for particular phases which are expected to have a low scalability e.g. due to collective communication. This is done to achieve power efficiency (reducing power usage when it is not beneficial for scalability), while improving performance.

Performance prediction is also needed in grid computing. [W.Smith 2010] describes the implementation of a queue wait time prediction service on TeraGrid [TeraGrid], based on runtime prediction. [F.Guim 2008] proposes a grid scheduler that depends on runtime predictions implemented as a service. [K.Kurowski 2005] mentions that grid resource brokers need estimations of job start time and job execution time to make decisions, rather than depending on simpler parameters like load. Similarly, [S.H.Jang, 2005] shows that selecting a site in a grid for execution based on performance prediction rather than using load information (i.e. assuming that the site with the lowest load will provide the shortest execution time), results in performance improvement. [U.Farooq 2009] presents a middleware framework for grids, which is capable of handling incorrect estimations of application runtimes, thus implying the potential benefits from accurate runtime estimations. [J.Zhai 2010] specifies that grid schedulers need estimations of individual workflow activities execution time to map workflow activities to different grid sites. [N.K.Kapoor 2010] describes matching resources to jobs using classes assigned to them according to their service demands; the proposed method is compared to one that requires a priori knowledge of jobs resource usage characteristics. [S.A.Jarvis 2006] presents two prediction-based middleware services and their usage to support the execution of a workload on a set of resources on grid. [F.Nadeem 2009] specifies the prediction of workflow execution time as having critical importance for optimization of workflow executions, and advance reservations of resources. [Nirav 1999] emphasizes the importance of runtime prediction in grid computing for resource management. [C.Glasnerlow 2011] specifies runtime prediction as a supporting service for schedulers used in grid computing.

The preceding discussion establishes a key role for performance prediction in various aspects of computing in general, and high performance and grid computing in particular. The rest of this chapter is organized as follows. Our taxonomy of different approaches and a survey of the state-of-the-art in performance prediction are presented in Section 2.1. Sections 2.2 through 2.5 provide details on each category of methods. We provide a list of key insights relevant to performance prediction in Section 2.6. Finally, our list of open problems and their importance is delivered in Section 2.7.

2.2. A Taxonomy of Performance Prediction

We consider the main aspect of distinction among performance prediction methods to be the level of abstraction at which they operate. In the literature, three different terms have been specified for these levels, which we will use as well throughout this report: black-box, white-box, and gray-box methods, as shown in Figure 1. We will describe each of the categories shown in Figure 1 in its own section, with subcategories shown in the figure described in the corresponding subsections. These differ in accuracy, cost and ease of use. The terms used in Figure 1 have been previously introduced by the literature, and organized by our taxonomy. At the lowest level of abstraction, white-box methods use information that is either only known by developers of the application or can be obtained through modification of application's source code or binary. These techniques consist of subcategories working at differing levels of abstraction. The main advantages of white-box approaches are their accuracy and the ability to answer what-if questions regarding performance. Their main disadvantage is the support they need in terms of developer/expert time, compiler/OS/tool support, etc, which makes them unsuitable for production environments. Black-box methods are on the other extreme, assuming only external knowledge regarding the application or platform, e.g. runtime and number of processes, usually obtained from logs of user activity across time/platform. The main advantage of this category is the potential for use in production environments, although some roadblocks, mainly the killing of jobs by

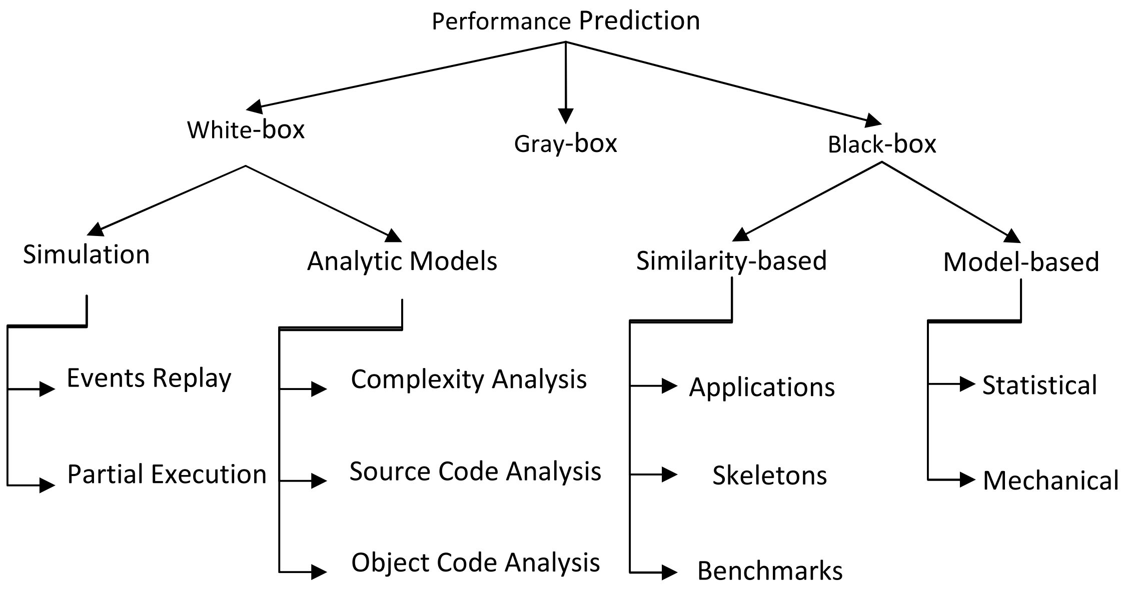


Figure 1. A taxonomy of performance prediction methods

schedulers due to underestimation of runtime, need to be resolved to actualize the wide applicability [D.Tsafrir 2007]. Gray-box methods operate at the middle abstraction level, attempting to maintain the applicability of black-box methods, while utilizing a subset of low-level information used by white-box methods, which is mainly problem size specified as a combination of input parameters.

2.3. *Black-Box Methods*

Black-box methods provide predictions without any “inside” information, i.e. only the external behavior of application is available, using two general approaches: 1) relating to behavior of “similar” applications/benchmarks, and 2) assuming a general behavior model, fine-tuned via model-fitting.

2.3.1. *Similarity to Other Applications*

The main idea here is that "similar" applications have reasonably close performances; hence if a set is formed of applications similar to the target, predictions can be made using observations on applications in this set [R.Gibbons 1997], [A.Downey 1997], and [W.Smith 1998]. The identification of similar applications requires the existence of historical information; these may be gathered in supercomputing centers, and there exists an archive of multiple centers logs [Parallel Workload Archive]. Most methods identify similarity on a per-site basis even if multiple logs are examined, probably because each center has its own unique set of applications and users. To identify similar jobs, [H.Li 2005], [W.Smith 2007], and [T.N.Minh 2010] use instance-based learning on jobs' attributes and [F.Guim 2008] constructs decision trees. In [W.Smith 1998] and [W.Smith 2004], sets of jobs' attributes, called similarity templates, are used to form groups of similar jobs. For example, the template (Username, N) places jobs with the same username and numbers of nodes in the same similarity group. Templates are determined using greedy and genetic algorithm search on a workload. The effectiveness of a template is related to measured mean error of the predictor fed the sets formed by applying the template to the workload. [K.Kurowski 2005] propose the GPRES expert system which also uses similarity templates but stores the extracted job-category-determination rules in a knowledge base. [F.Nadeem 2009] constructs similarity templates using supervised exhaustive search on grid workflow-level attributes, e.g. set of activities, application-level attributes, e.g. problem size, execution-level attributes, e.g. set of grid sites, and resource-level attributes, e.g. jobs in the queue. [C.Glasnerlow 2011] uses a set of similarity rules (e.g. jobs submitted between 8am to 4pm are similar) and the resulting clusters of a single user's jobs, which are assigned relevance for a particular job type based on accuracies in previous predictions. [S.Krishnaswamy 2004] Identifies similar jobs using rough set theory, where job characteristics and performance are condition and decision attributes, respectively, forming the similarity templates based on dependence degree of decision attribute on each condition attribute. [R.Duan 2009] Uses specially-structured Bayesian networks where factors are job attributes and correlation coefficients is used to discard irrelevant factors. The probability distributions between factors are calculated from the observations dataset.

The next step is to derive a prediction from observations on the set of similar applications. [D.Tsafrir 2007] reports improved scheduler performance when taking the average of the similar jobs from the user's history, with higher accuracy from more recent and less similar jobs than otherwise. [D.Talby, 2006] introduces a session-based history (SBH) predictor, sessions being sets of an individual user's jobs with at most 20 minutes between termination of one and submission of next, which uses the median of similar jobs across multiple sessions. This is compared to recent user history (RUH) predictor, which uses the median runtime of the last 3 terminated jobs of the user, showing slightly higher accuracy for the former. As a result of experimenting with different configurations of SBH, authors report improved results from using exact but farther in the past (up to 30 sessions) matches versus using partial but more recent ones, i.e. exact similarity is more important than proximity in time. In the extreme, considering only the most recent session and ignoring similarity performed even worse than RUH. [F.Nadeem 2009] uses the average of similar jobs, with the possibility of shifting the prediction toward more recent items versus using all available observations. [S.Krishnaswamy 2004] and [T.N.Minh 2010] use the mean runtime of the set of similar jobs. However, considering the context of the predictor in [T.N.Minh 2010], i.e. backfilling scheduler, the number of underestimations is reduced by adding a fraction of the standard deviation of K neighbors' runtime to the estimation, and using the user-provided runtime as the upper-bound for the estimate. In [R.Gibbons 1997], author proposes a method that uses averages and provides confidence intervals; however, formation of sets of jobs for average calculation is not specified. In [W.Smith 1998] and [W.Smith 2004], runtime prediction and its associated confidence interval are obtained from multiple sets of similar jobs (called categories) by either calculating the mean of runtimes or using linear regression with the number of nodes as the regression variable. There is also a maximum on the number of jobs in each set, and the oldest job in the set is discarded if that maximum is passed due to addition of a new job. If the target job falls into several sets, a prediction is made per set and the prediction with the smallest confidence interval is selected. In [W.Smith 2007], a prediction and a confidence interval are obtained using a kernel regression method applied to the N observations that are most similar to the target application, called query.

Genetic algorithm is used to search for optimum configuration of the regression method, e.g. kernel function width, feature weights. ADAPS, proposed by [C.Glasnerlow 2011], uses multiple prediction methods applied to multiple sets of jobs formed based on similarity. To make a prediction, job sets (active clusters) to be included are selected and weighted average is taken among the predictions made per pair of job set and method, where both the selection of job sets and weight assignments use accuracy feedbacks. [F.Guim 2008] uses the C4.5 decision tree algorithm, which results in prediction of ranges of runtime rather than point values. In [R.Duan 2009], authors use an RBF-NN (radial basis function neural network), fed by a Bayesian network. The Bayesian network provides the RBF-NN with a reduced number of dimensions and probability tables (e.g. the probability that runtime is between 980s and 1080s when the preparation time job attribute is between 0s and 215s). In [K.Kurowski 2005], authors propose a method based on similarity rules from a knowledge base, which uses the arithmetic mean of the result variables of two target application-matching rules: the one with highest specificity and the one with highest number of matching jobs. In [H.Li 2005], authors use instance-based learning, either 1-NN or N-weighted averaging, finetuning the parameters of the predictor using a genetic algorithm search, where fitness is based on prediction accuracy on training dataset,

2.3.2. *Similarity in Terms of Benchmarks and Hardware Metrics*

Another group of black-box methods attempt to predict by relating the performance of target to that of benchmarks, usually based on hardware-level metrics as these maintain the black-box constraint. [S.Sharkawi 2009] proposes a method for predicting cross-platform, node-level performance, i.e. communication ignored, of constant working-set size HPC applications by relating to SPEC CFP2006 benchmarks. The overall approach is to use a genetic algorithm tool to derive a performance model for the application as a weighted combination of similar benchmarks (called surrogates), via examining relative contributions of 6 groups of hardware counter metrics, obtained on a base machine, at both inter and intra group levels. Performance is predicted by combining the model with the published performance data of benchmarks on the target platform via solving the set of linear equations resulting from the latter, yielding the platform-specific function H which relates runtime to benchmarks using 6 coefficients.

In [W.Pfeiffer 2008], Pfeiffer et. al. model the application runtime as the weighted sum of published machine characteristics and measurements made by HPC challenge micro-kernels (e.g. Peak flop, interconnect latency, memory bandwidth), with weights being platform-independent application coefficients calculated by model fitting on its runtimes across different platforms and numbers of cores. To make predictions for a platform, the model is combined with the measurements made by HPCC benchmarks on that platform. The method addresses robustness and goodness of the fit by checking for outliers and influential measurements, i.e. single measurements the elimination of which significantly changes the fit. Backward elimination is employed to allow only statistically significant predictors in the model. Communication time and fractions are gathered and a larger dataset for the fits is obtained by separately specifying communication time as a function of related predictors. Authors suggest 15 to 20 measurements to fit three or four parameters for a given benchmark. In [F.Freitag 2001], Freitag et. al. propose a low-overhead speedup prediction method for a hybrid application, i.e. one that uses a combination of MPI and OpenMP, via dynamic detection of its iterative structure and parallel loops through monitoring the changing CPU usage.

Thomas et. al. introduce a profiling and performance analysis tool for MPI applications, which does not require re-compiling or re-linking the target application to obtain communication traces [D.Thomas 2010]. The tool can identify wait times due to both collective operations and delay between send and receive operations. The tool is then combined with hardware counter information to provide runtime estimates for parallel applications.

2.3.3. *Similarity: The Concept of Skeletons*

In [S.Sodhi, 2008], Sodhi et. al. propose a similarity-based approach that constructs a performance “skeleton” of the application: a synthetic, orders of magnitude shorter program with a runtime that is a fixed portion of that of the application, under any scenario/platform. The proposed method automatically constructs skeletons via identifying and summarizing repeated patterns, i.e. segments of similar system activity, in the application’s execution trace, leading to an execution signature that is transformed into the skeleton. To obtain the signature, similar MPI calls are identified and represented with the same symbols, transforming the trace into a string, which is compressed into a loop structure by recognition of repeated patterns. A synthetic program that is representative of the signature, i.e. has similar execution trace, is then constructed, using the identified loop structure to scale down the runtime. Execution trace is identified by using the standard PMPI interface to link the application with a profiling library. The computation time is calculated as the time between the call and return of MPI routines, i.e. suffers the same problem as most other work in terms of specifying the end of communication as the return time of the MPI call. The method is currently more suitable for performance prediction under load sharing and not across different platforms, i.e. different CPU and interconnects architectures. Computation is only briefly specified, and memory subsystem behavior and its role in prediction is skipped.

In [A.Toomula 2004] (mostly by the same group), a method is proposed for constructing a skeleton program which has the same cache behavior, in terms of number of cache misses, as its target application, on any platform. Because the collection of all memory references of the target application is impractical, samples of memory references are collected, using Valgrind tool [Valgrind], each sample being a sequence of memory references long enough to capture temporal locality. The references are stored as the number of the cache line they access, are clustered and used to generate the skeleton's synthetic C program. In [Q.Xu 2008] (from mostly the same group), a method is proposed that constructs skeletons by combining traces from multiple processes into a logical trace. In addition, authors specify the use of synthetic computation code which is the same in duration, but does not entail the memory behavior of the target application.

2.3.4. *Black-Box Methods Using Mathematical and Statistical Models*

A subcategory of black-box methods assumes general conformity of target applications to an underlying model with coefficients determined for each application based on observations of its behavior.

In [H.A.Sanjay, 2008], application runtime is specified as an equation that depends on functions of communication and computation complexity, amount of parallelism in computation and communication, and CPU and network loads. Linear regression is used to make predictions, and different functions are selected based on the current load of CPU and network. A set of at most 20 candidate complexity functions are determined by running the target application for different problem sizes on a single non-dedicated CPU (for computation complexity), and on two non-dedicated CPUs (for communication complexity), fitting the set of all potentially relevant complexity functions to the observations, and choosing the functions with smallest fitting errors. Scalability is modeled through functions specifying amount of parallelism in computation and communication, via running the application on 2, 4, and 8 processors. The overall obtained model is used to predict runtime under various values of loads, number of CPUs, etc. In [R.Wu 2008], a pure mathematical approach specifies the runtime of a parallel application as the maximum runtime of its processes and individually models each process as a Johnson distribution. Tudor and Teo provide an analytical model for speedup for shared-memory programs on multi-core systems, which uses hardware counters and operating system run-queue [B.M.Tudor 2011]. The model measures the number of cycles lost to memory contention and data dependency, and calculates an estimated speedup loss due to these cycles. The proposed model is evaluated on 6 OpenMP HPC dwarfs from the NAS benchmark suite.

2.3.5. *Black-Box Methods Using Mechanical Models*

Black-box models may use non-statistical models, which are based on certain characteristics of parallel applications, and have been called mechanical models in the literature. In [S.Shimizu 2009], Shimizu et. al. model resource consumption statistics, in particular the runtime, of a specific problem size of the application as products of resource-specific terms including contention, e.g. $ExecTme = (a_{00} + a_{01} * Z_{clock})(a_{10} + a_{11} * Z_{cache})$, with the coefficients obtained by applying regression analysis to observations of application across platforms. The model is claimed to improve in accuracy as the number and variety of platforms are increased. In [S.Venkataramaiah 2003], authors model the performance of a specific problem size of the application as a function of application's behavior and level of contention over CPU and interconnect. The parallel application is run on a dedicated platform, and measurements of CPU usage and network usage are combined with reassembly of application's messages and platform benchmarking results to determine the time each CPU spends on computation, synchronization (wait), and communication, used to predict the performance under different contention levels. CPU usage is monitored via CPU probing, and tcpdump provides network traffic logs. In [A.Deshmeh 2010], the ADEPT predictor is proposed which uses the Downey model [A.Downey 1997_2] as the underlying model that explains the behavior of parallel applications. Observations of target application's execution times for the same problem size over different numbers of processors are used for model fitting. A separate model fitting is done per prediction target, assigning weights to observations based on their distance from the target prediction point. ADEPT also handles individual anomalies in the observations by introducing a novel heuristic that is based on expected scalability of a parallel application. Anomalous behavioral patterns, e.g. major runtime improvements at processor counts which are powers to two, are also handled via introduction of multi-phase modeling. ADEPT is also capable of detecting unreliable predictions, e.g. when significantly distinct instances of the model can be fitted to the existing observations.

2.4. White-Box Methods

We divide white-box methods into two general categories: a) analytic modeling which is composed of model-driven techniques; these are distinguished by the abstraction level(s) at which the required analysis is performed: complexity level, source code level, and object code level, and b) simulation, which covers techniques that base the prediction on some mimic of the application's execution; these too are distinguished based on their abstraction level: overall structure-level abstraction leads to partial execution, while instruction-level abstraction defines event replay techniques. There is some degree of overlap among the two general categories, which we will point out.

2.4.1. *Analytical Modeling*

At the highest level of abstraction among analytical modeling techniques are those that derive a performance model by analyzing an algorithm rather than an application. In [K.J.Barker 2009], Barker et. al. construct a model of a generic five-point stencil application by analyzing the general algorithm that the application follows. In [J.Schopf 1998], Schopf et. al. analyze the performance of a stencil application, but specify the parameters of the model as distributions rather than single values. In [M.M.Mathis 2005] Mathis et. al. use complexity analysis to construct a model of mesh particle transport computations. Such methods provide an overall expectation of the performance of a specific solution regardless of the implementation details; they, however, run the risk of ignoring factors that critically influence the performance, e.g. cache attributes of the target platform.

The next level of abstraction entails methods that employ source code analysis. In [A.V.Germund 2003], Germund et. al. separately model application and platform and combine the two to obtain a symbolic model for performance prediction. Authors provide mechanisms for translating specific parallelism patterns into models. They also provide detailed methods and discussions for transforming different programming constructs, e.g. pipelining, and phenomena, e.g. memory and network contention, into the proposed modeling language. The PACE toolkit [G.R.Nudd 2000] separately models the application and the platform, and combines the models to obtain performance predictions. The toolkit is able to predict performance for different numbers of processors. In [M.M.Mathis 2006], Mathis et. al. also separately model the application and platform, using a modified version of the CHIPS performance specification language [G.R.Nudd 2000]. A CHIPS model has a hardware specification component and a task graph representation of the parallel application based on detailed knowledge of source code. The proposed method predicts the time required per cell, processing unit of the application, for different cells per processor.

In [S.R.Alam 2006], Alam et. al. propose a method for predicting workload and memory requirements based on an API for MPI programs in FORTRAN and C, which generates trace files that contain key events e.g. communication events, loop start/end, floating point operations start/end. The constructed model is based on computation, communication, and key input parameters of the application. In [L. Adhianto 2006], Adhianto et. al. propose a prediction method that addresses hybrid applications (MPI + OpenMP), which uses the compiler to obtain an application signature consisting mainly of memory access patterns and floating-point operations. The method uses benchmarks to obtain platform characteristics, e.g. cache size, cache line size, clock speed, and the parallelism overhead of MPI and OpenMP. [M.Nakazawa 2005] uses performance prediction to find the best data distribution for a parallel application. It addresses I/O cost as well as computation and communication cost for building a model of parallel programs in terms of a set of equations. Micro-benchmarks are used to identify the initialization cost, send and receive overheads, etc. The method assumes that parallel applications are iterative, and measures the time for instrumented run of one iteration of the main loop to obtain computation, communication, and I/O cost. Manual analysis of the source code is required to identify parallel sections, which are then instrumented to obtain their computation time and I/O time. The computation times for different amounts of work are obtained using these measurements.

In [Z.Wang 2009], authors propose a machine learning-based method to determine the best number of threads for an OpenMP program on multicore machines, based on prediction of the scalability curve of the program. A neural network and a support vector machine are trained off-line on features extracted from a set of programs, and are fed the features of new programs to output the optimum number of threads and scheduling policy. The features that are to be provided on both training set programs and the new program are extracted from both the source code, e.g. load/store, branch count, and dynamically using source code instrumentation, e.g. L1 data cache miss rate. [T.Fahringer 2000] proposes a performance prediction framework which uses the source code written in HPF and instrumentation to obtain a model of the parallel application based on work distribution, communication parameters, cache misses, and computation time. To predict computation time, it uses the runtime of kernels executed on the target architecture. [J.Li 2009] introduce a method that uses neural networks for predicting execution time of functions (tasks) of an application using its source code; the input to the neural network is the previous runtimes and input parameters. The neural network then predicts execution time and size of output (since it affects cost of communication between tasks) for the function. The application needs to be written in a language called R script to be processed by the predictor.

There exist varying levels of abstraction among methods that depend on source code analysis. Methods at higher levels of abstraction specify the application's runtime as a function of problem size, data distribution, etc. This usually allows only the implicit inclusion of platform characteristics and effects. Methods at lower levels of abstraction, e.g. [G.Marin, 2007] specify the control flow graph of the application and identifies the basic blocks and the set of operations they perform. These allow the explicit consideration of platform capabilities, e.g. specifying how much time each basic block needs on a particular platform based on the block's needs and platform's resources, at the cost of more expert time and potentially the added requirement of instrumenting the runs to obtain some of the required metrics, e.g. number of floating-point operations of a basic block.

Methods in the final group of analytical modeling analyze the object code/executable of the target application, via instrumentation. A typical example of using instrumentation is obtaining communication characteristics of applications, e.g. how many bytes are sent on average per process, what is the average message length, what percent of the communication operations are collective and thus may involve long waits, etc. As in code analysis, the resulting application model may need to be combined with a platform description to provide performance predictions. Instrumentation of applications' binary or source code attempts to automate at least some parts of the code analysis to reduce the time required of a performance specialist or the developer or to replace them; the latter may result in some sacrifice in terms of accuracy of the constructed model.

In the Prophesy project, [V.Taylor 2003] specify the main innovation to be the automatic modeling component. The main measure is the coupling parameter that specifies the interaction among kernels that make up an application. A kernel is a logical unit of work; it may be a loop, a file or a procedure. The coupling value between two kernels is the result of the division of their consecutive execution by the sum of their individual execution (measurements for each of these terms is done in the form of a loop execution either an individual kernel or a chain of 2 or more kernels) [X.Wu 2004]. The data collection component of the Prophesy framework collects data using automatic instrumentation at the basic blocks, procedures, or loops level. The modeling component provides three methods: curve-fitting, parameterization, and kernel coupling. In curve-fitting, the user selects the data and the method to use, and models the application runtime, communication performance, etc. as a function of some or all of the input parameters of the application. The parameterization method involves hand-counting the number of different operations in the code, and grouping them to construct formulas which contain coefficient that can be determined from the database using hand-written scripts. The Valerie Taylor group has used the Prophesy project for runtime prediction on different HPC applications [X.Wu 2006_1], [X.Wu 2006_2]. The kernel coupling measures the interaction between kernels by dividing the combined runtime of kernels (runtime of kernels when they are run in sequence) by the sum of their individual runtimes. These runtimes are measured by placing one or more kernels into a loop such that the loop dominates the runtime, measuring the new runtime, and subtracting the time required for execution of the rest of the application from the obtained runtime. The application runtime is then modeled as the summation of kernel models (kernel models seem to be developed using the parameterization method, multiplied by the number of times it is executed in the application), each multiplied by a coefficient which is calculated as a linear function of kernel coupling values.

[V.Taylor 2001] proposes a method to automate the development of analytical models of parallel and distributed applications. Data about an application are gathered via instrumentation and stored in an application performance database (also the compilers, libraries, and the control flow). There is also a model template database, and a systems characteristics database. The goal is to use the three databases to make predictions on performance of an application on different system configurations. Specifies three modeling methods: curve-fitting, parameterization (these two are also specified in other Prophesy papers), and composition. The argument for parameterization which is manual is that parallel applications are composed of a few key kernels and it would suffice to focus on these kernels. The composition method seems to be the same as kernel coupling method specified in later relevant papers. Runtime prediction is done for an example application (matrix multiplication), and an FFT benchmark from NPB suite. According to [V.Taylor 2002] kernel coupling values are weighted based on the fraction of the runtime attributed to the corresponding kernels. It should be noted that kernel coupling values can be generated in a pair-wise manner (i.e. for pairs of kernels) or for chains of 3 or more kernels. The exploration of which number of kernels in the chains leads to better results is left for future work.

He, et. al., propose a method for identification of data flow patterns in parallel programs, e.g. reduction, which can be used for performance prediction [J.He 2011]. The source code of the application is used in static analysis to classify the data flow as one of the 5 defined patterns. The loop nest structure is extracted from the intermediate presentation of the code prepared by the compiler, and all the assignments are examined to construct a graph relating the result to the program variables. A reduced form of this graph is then compared against the predefined graphs for recognition of data flow pattern. Authors then relate the performance of several synthetic benchmarks to those of NAS benchmarks, by matching the data flow patterns.

[L.Carrington 2003] independently models both computation and communication operations of parallel applications (called application signature) and machines (called machine profiles), and convolves the two models (separately for computation in terms of a single processor model: memory and floating-point operation needs / corresponding machine rates, and communication) to predict application runtime on a specific machine. The machine profile is composed of machine's capability to perform certain operations, e.g. peak floating point rate, obtained via low level benchmarks called probes. Performance of 3 scientific applications is modeled, with generally below 20% runtime prediction errors. The paper also provides a discussion on using the model for sensitivity study (e.g. what would be the performance of the application if network bandwidth was doubled, etc.).

Cornea et. al. demonstrate the use of a performance prediction tool called dPerf on applications in P2PDC environment for high performance P2P computing [B.F.Cornea 2011]. The dPerf tool combines static and dynamic analysis with simulation of the obtained traces for prediction. The source code of the program is used in the static analysis to obtain basic blocks, and instrumentation provides the runtime of each block. This data is then fed to a trace-based network simulator to obtain an estimation of the runtime of the parallel application.

2.4.2. *White-Box Methods Using Simulation*

Simulation-based methods provide performance prediction of a target application by mimicking its behavior on a platform. The input to simulation is a representation of target application's behavior, which can include a full event trace of the application covering categorization of different operations, e.g. floating-point, obtained using instrumentation [M.Tikir 2009], [L.Carrington 2005], [G.Marin 2007], or only the communication events obtained via linking the application with a profiling library and recording all the communication calls made by the application [G.Rodriguez 2004], [M.Casas 2008]. It should be noted that there is some overlap between simulation and analytic modeling approaches, in terms of the constructed analytical model being used by a simulator, of e.g. the target platform, to provide performance predictions.

The first subcategory of simulation-based methods entails techniques that record events occurring during the execution of the target application, and replay these for performance prediction. [M.Tikir 2009] Collects events during an application's run, and is able to replay and simulate these traces (to model current and future HPC systems). The tracer is built on MPI's profiling interface. The time between communication calls, called CPU bursts, are also recorded. The simulator takes as input the event trace for an application, and a set of configuration parameters for a target system (parameters of the global system, each compute node, and the task-to-processor mapping; CPU speed is specified as the ratio of target to base system), and simulates the execution of the application on target system. Computation time is estimated by multiplying CPU bursts by the ratio of CPU speed between target and base systems. In simulation, each event is labeled with its earliest ready time. Communication models are separated from the simulator, and use the configuration files and the current state of the system to calculate the sustained latency and bandwidth, and decide when a particular event will be executed, e.g. depending on the availability of the resources in the communication system.

In [L.Carrington 2005], Carrington et. al. support the idea of relating HPC applications' performance to simple benchmarks via a runtime modeling and prediction framework, which captures the applications memory and communication characteristics via traces. Examines the accuracy of a simple prediction method, T'(x,y)= (R(x) / R(x0)) * T(x0, y); x0 is the base system, R(x): simple benchmark on system x, T(x,y) : runtime of application y on machine x, to conclude its insufficient accuracy. Proposes a predictive framework in which applications' operations are divided into categories, and instrumentation is used to gather the count for each operation for an application (e.g. number of floating point operations). The MetaSim Convolver [A.Snavely 2003] is used to combine operating counts and operation rates, which are obtained via simple benchmarks. The time of these categories of operations are then summed up to predict the applications runtime, taking into account the overlap between operations.

In [G.Marin, 2007], Marin et. al. use static (to obtain control flow graph) and dynamic (memory usage patterns, etc.) analysis of object code to develop a model of the parallel application. The application model is combined with the machine model (architecture description) to predict runtime, using a module instruction scheduler that maps application operations on resources of the target machine. Addresses cross-platform and different input parameters (using models parameterized by input parameters). A machine description language is provided for describing different platforms.

[M.Casas 2008] Instrumentation is used to obtain data on MPI calls of the application. A set of parameters: communication efficiency, load balance, average IPC, and number of instructions, are defined and measured for different numbers of processors. The values of these parameters are then related to number of processors using log-linear fitting, and an analytical model based on these parameters is then used to predict runtime for larger numbers of processors. For performance prediction on different interconnects, the bandwidth and latency, as well as network topology are taken into account using the Dimemas simulator [Dimemas 1997].

[G.Rodriguez 2004] proposes a linear model of parallel applications, which is based on critical path length, number of exchanged bytes, and number of non-overlapped latencies. The method uses dynamic instrumentation to obtain communication requests and CPU demands for different numbers of processors, and feeds these to Dimemas [Dimemas 1997] simulator. Regression is used on simulation results to fit the model to the application. Validation has only been done for simulations by Dimemas and not actual runs. [S.Pllana 2005] Uses source code of a parallel program to group the statements into categories like computation, loop, send, receive, and barrier. Also uses a simple model of machine: number of nodes and number of cores per node, etc. The execution of the modeled program on the modeled machine is then simulated to obtain a performance prediction. The program is first modeled using UML, and the UML model is automatically translated into a performance model. [J.Zhai 2010] Clusters processes of a parallel program into groups with similar behavior, runs one representative from each group on a single node to model the computation time, and combines the real sequential computation time measurements with a trace-driven network simulator. Deterministic replay is used to allow the execution of a single process of an application on a single node.

In [S.Achour 2011], Achour et. al. present a framework for prediction of parallel application’s performance, which uses regression to profile the computation kernel and communication of the parallel application. The framework feeds the models obtained on computation and communication of the target application to a simulator to obtain runtime predictions. The modeling assumes availability of the source code of target application in C language, and collects traces of both computation and communication to be provided to the simulator. The simulator constructs the task graph of the target application and predicts the execution time of each task, and calculates a runtime estimation by addressing wait times in addition to tasks runtime.

It should be noted that there is some degree of overlap between analytical modeling methods and event replay methods as a subcategory of simulation-based methods; e.g. [M.Casas 2008] uses a model that is based on several metrics, and [G.Rodriguez 2004] employs a linear model.

A final group of white-box methods use partial execution for performance prediction. The argument for this approach is the intrinsic repetitiveness of parallel applications, which means that after an initial startup period, the parallel application goes through a loop and each of the iterations of the loop demonstrates similar characteristic, including runtime, to others. This category of methods thus attempts to extract, as the model of the target application, the set of operations which are done repeatedly. This model can then be used to measure the performance on the target platform, at a cost which can be orders of magnitude less than the cost of running the target application itself.

[L.T.Yang 2005] argues that parallel applications are iterative after a startup period, and thus partial execution can be used for performance prediction. The performance of a parallel application on a target platform is predicted based on its performance on a base platform, and the relative performance of the two platforms obtained via partial execution. For the approach to work the introduced API needs to be used by the source code, i.e. source code modifications are required. Either the number of time steps or a full execution of application on base platform needs to be known. Communication is ignored. The limitations of the model are specified as not addressing different input parameters or different numbers of processors (i.e. no scalability prediction). [J.Corbalan 2005] mentions a runtime library called SelfAnalyzer, which measures speedup and predicts runtime of parallel applications. The tool depends on internal structure of parallel applications, in particular the main loop. It runs several iterations of the main loop on a small number of processors, called baseline, and from then on runs the iterations of the main loop on the requested number of processors. This runtime is used to calculate the speedup versus baseline. If the source code is not available, instrumentation is used to inject SelfAnalyzer code into the target application. The tool currently runs on OpenMP jobs which are malleable, and not MPI jobs. The analyzer is mostly focused on speedup, not runtime prediction.

2.5. Gray-Box Methods

Gray-box methods are a more recent approach to performance prediction: the term was introduced by [B.Barnes 2010], although older examples of the approach do exist: [E.Ipek 2005], [B.Lafreniere 2005]. The general idea is to employ elements from white-box methods thus approaching their accuracy, while minimizing such usage so as to maintain a cost and applicability close to that of black-box methods. These methods generally perform model fitting on the problem sizes as points in the input parameters' space; thus, our analysis differentiates them based on the properties of the fitting.

In [B.Barnes 2010], Barnes et. al. propose a method that uses similarity in the parameter space to predict parameter values that would result in time-constrained scaling, i.e. increasing problem size to maintain constant performance on increasing numbers of processors. Performance for a problem size is predicted using "focal regions" of the parameter space, which represent smaller problem sizes but have similar ratios of input parameters (e.g. for the problem size specified using the parameter triplet (1,32,32), focal region includes (1, 16, 16), (1, 8, 8), but not (1, 8, 32) or (1,4,32)). Fitting of a log-based model relates the execution time, and separately communication time if it is significant, to computation time and number of processors. Training data is the performance observations at different points in the parameters space, assuming knowledge of time-step loop to minimize the cost of obtaining observations and using at most half the target number of processors. [B.Barnes 2008] provides scalability prediction for strong scaling, in which increasing number of processors reduces runtime for a constant problem size, via extrapolation in the parameters space: points in the parameter space with small numbers of processors are used to predict runtime on a large number of processors. The method separately relates computation and communication time, assumed as non-overlapping, to parameters and a function of the number of processors through log-based regression. Communication time is measured using PMPI profiling interface; one variation of the approach uses global critical path to exclude blocking time.

In [B.C.Lee 2007], Lee et. al. propose performance prediction using parameter-based models using either piecewise polynomial regression or neural networks. The selection of predictors (characteristics of application or processor grid) to include in the models is guided using statistical methods. Hierarchical clustering is used to classify predictors into highly correlated groups, and the significance of predictors is quantified using correlation analysis. Either uniform random or regional sampling, the latter based on similarity to the query, is used compose the training set of data points in the parameters space. In [E.Ipek 2005], Ipek et. al. train neural networks on the space of input parameters and the resulting performance values. The parameter space is sampled using regularly spread points (runtime for these points is actually measured). In [B.Lafreniere 2005], Lafreniere et. al. propose a method that depends on user-specified "rough" linear formula to relate performance to application's characteristics and input parameters. Model's coefficients are determines using regression over a dataset of performance versus independent variables. In [A.Matsunaga 2010], Matsunaga et. al. construct a decision tree in the space of input parameters and platform characteristics, e.g. CPU architecture and memory size and speed, to predict resource usage, runtime in particular. At the leaf level, regression is used with finer granularity, i.e. the leaf determines the performance range and the regression method makes a prediction within this range. The idea is to select, from a pool of methods, the best regression method for each set of data. The proposed method is evaluated on two bioinformatics applications on different platforms, concluding that different machine learning techniques may be appropriate at different situations, hence a need for adaptive methods. In [Nirav 1999], authors relate the runtime to the input parameters using K nearest neighbors, K nearest neighbors with weighted averaging (weights are the reverse of distance of the neighbor from the target point), and locally weighted polynomial regression. A knowledge base and caching of results are used to reduce the overhead of the prediction scheme. [F.Nadeem 2006] introduces G-Prophet, a system for cross-platform performance prediction, which employs linear regression and uses a performance-translation mechanism to provide a larger training dataset at a lower cost. The mechanism assumes that the performance ratio for a base problem size to that of any other problem size is constant across all grid sites. Thus, results from running one base problem size on selected grid sites are combined with those of running all problem sizes on the fastest site to provide the training dataset. To further reduce training cost, the method forms sets of similar grid sites, i.e. same number and architecture of processors, memory size and characteristics, and OS, and uses one site from each set as a representative.

2.6. *Evaluation Methods and Applications of Performance Prediction*

Comparing predicted and actual performance for one or more target applications is the most typical method in evaluating a performance prediction method. [A.Matsunaga 2010] experiments with Basic Local Alignment Search Tool (BLAST) and Randomized Axelerated Maximum Likelihood (RAxML). [S.Sodhi, 2008] uses class B of NAS benchmarks for the experiments. [S.Venkataramaiah 2003] also uses NAS benchmarks. In [M.Casas 2008], NAS benchmarks BT, SP, and MG have been used for the experiments, but the class of benchmarks is not specified. [G.Rodriguez 2004] evaluates the proposed method on NAS BT, Sweep3D, RNAfold and POP [POP Application] application. [M.Nakazawa 2005] uses CG NAS benchmark, Jacobi Iteration, RNA pseudoknots [L.Cai 2003], and Lanzcos iterative method. In [M.Tikir 2009], experiments are performed on three scientific applications, ranging from 0.5 to 2.5 hours in runtime. [L.Carrington 2005] uses 5 real-world HPC applications. [H.A.Sanjay, 2008] experiments on several parallel applications, e.g. ScaLAPACK eigen value solver and integer sort (IS, but not part of NPB). [R.Duan 2009] uses execution traces of real grid workflow applications. [S.Pllana 2005] experiments on a single program: LAPW0. [H.Li 2005] uses logs of NIKHEF cluster as the testing dataset. [B.C.Lee 2007] targets 2 applications: SMG2000 and HPL [A.Petitet]. [E.Ipek 2005] uses SMG2000 code. [S.Krishnaswamy 2004] uses the some data mining applications to evaluate the performance of the proposed method. Although our list is not exhaustive, it demonstrates the variety of target applications and a lack of a generally accepted set of “representative” applications which has complicated the comparison of performance prediction methods.

Another common approach for evaluating a performance prediction method is to use the logs of jobs executed in a supercomputer center, or, similarly, evaluate the changes in the performance of a scheduler that employs the proposed prediction method over such logs. [W.Smith 2007] uses machine logs from two months as the training dataset, and machine logs of a different month as the testing dataset, thus ignoring potential locality in the logs. [S.Krishnaswamy 2004] additionally uses the San Diego supercomputer center 1995 log [SDSC95] and the San Diego supercomputer center 1996 log [SDSC96]. [F.Guim 2008] evaluates the benefits of runtime prediction using simulation of the scheduler, but does not compare the effect of using other prediction methods on the same metrics. [F.Guim 2007] presents experiments showing the effect, on scheduler performance, of varying levels of errors in different categories of jobs, for both quantitative and qualitative errors (e.g. predicting a long job as short, etc.). Regarding the former, it is concluded that highly accurate prediction of runtime for short jobs is crucial to performance of scheduler, whereas a higher prediction error is acceptable for long jobs. As for qualitative errors, a high impact on scheduler performance is reported (exponential tendency on the average bounded slowdown) if qualitative errors are made by the predictor, particularly if it can happen in both directions of predicting short jobs as long and vice a versa. It is thus advised that any prediction method should attempt to avoid such errors, recommending the provision of confidence intervals as a possible solution. However, the only methods that provide confidence intervals are [W.Smith 1998], [W.Smith 2004], [W.Smith 2007].

[E.Shmueli 2009] specifies that dependence on a predetermined workload for examining the performance of a scheduler is unrealistic. Instead, authors propose user models to simulate the behavior of users in submitting jobs, taking into account that the behavior of the user is influenced by the scheduler. More specifically, user actions can be grouped into sessions, which are sets of job submissions separated by short “think times”. Authors claim that a “better” scheduler is one that encourages users to continuously submit more jobs, by addressing criticality of jobs from the users' perspective and not arrival order alone. Also in [E.Shmueli 2006], authors propose a detailed model of users' behavior which is claimed to be more realistic due to addressing the impact of scheduler's decisions, and thus more suitable to evaluate performance prediction methods. Through experiments, it is demonstrated that evaluating the performance of one scheduler using trace data obtained as the response of users to another scheduler can result in significant underestimation or overestimation of performance metrics.

The existence of anomalies in the test set can significantly affect the judgment of the effectiveness of a prediction method, particularly if the evaluation is done in the context of a scheduler. [D.Tsafrir, 2006] proposes a method for detecting and eliminating anomalies in the workloads, used later by [F.Guim 2008]. [D.G.Feitelson, 2008] also emphasizes the need to clean platform logs from abnormal activity: an example is shown in which cleaning the abnormal activities of one user from a machine log leads to significant change of the calculated correlation between runtimes and job sizes. [C.Glasnerlow 2011] uses an outlier detection mechanism that considers the last completed job an outlier if it does not conform to previous ones. [A.Deshmeh 2010], as explained in black-box methods section, uses a fluctuation metric which is based on expected scalability of the target application to identify both individual anomalies and those that are part of a specific scalability pattern, e.g. an application that runs well only on processor counts that are powers of two.

In addition to using a variety of target applications to evaluate performance prediction methods, reporting of the achieved accuracies is also done in various methods, further complicating the comparison of performance prediction methods. In [W.Smith 1998], results are compared to other methods, showing smaller average mean error for the proposed method, measured in minutes. The mean error, also measured as the fraction of mean runtime, is reported to be between 42% and 70%. [W.Smith 1998] and [W.Smith 2004] report mean prediction errors between 29 and 59 percent of mean application run times. In [W.Smith 2007], runtime prediction errors are reported as 72% of the mean execution time and compared to user runtime estimates errors of 246%. The overhead of making predictions for a particular platform is also specified. In [S.Krishnaswamy 2004], Krishnaswamy et. al. report mean errors as percentages of mean runtime. A major part of the experiments are performed on the SDSC data, with test cases obtained randomly from the log. In [F.Guim 2008], due to predictions being ranges of runtime rather than point values, it is not easy to compare to other methods as the mean error in terms of runtime is not specified (only 160% average error and -1.7% median error are mentioned, but calculation base is missing, which is probably categories of runtimes rather than actual runtimes). In [M.Tikir 2009], prediction of communication time based on simulation is reported to have an error of around 14%. Highly accurate predictions are also reported for runtime prediction using simulation, with generally less than 20% errors.

2.7. *Key Insights Provided by the Literature*

Next we describe a set of insights, provided by the literature either in direct association with performance prediction or otherwise, which we consider to have significant implications for performance prediction.

2.7.1. *Job's Size, Runtime, and Potential Correlations*

There have been studies attempting to establish relationships, e.g. correlations, between job sizes and other job attributes, mainly runtimes, through examination of logs from supercomputer centers. A key implication of a strong relationship, as noted by [D.G.Feitelson, 2008], would be that scheduling decisions are implicitly based on runtime, due to dependence on job sizes. [D.G.Feitelson, 2008] shows the different percentages of the jobs with different sizes across several supercomputing centers, demonstrating a strong preference for powers of two sizes. The experiments do not show a strong or even uniform-across-all-logs correlation between job size and runtime, although categorizing jobs into small and large categories showed a stronger but still inconclusive, i.e. not uniform across logs, correlation. [E.Shmueli 2009] also performs a similar study using the CDF of job runtimes and sizes, specifying the consideration of the correlation between size and runtime as a means to gain further accuracy in simulating user behavior. [U.Liblin 2003] models runtime and size as a combination using the correlation between the two, reported as the observation of two gamma distributions for the runtime of each of the 3 size-based categories of jobs. A hyper-gamma distribution models the runtime per category, with a size-based parameter p specifying the distribution to sample. The paper also reports a much higher correlation between runtime and job size for batch jobs than for interactive jobs, and the peak of runtime distribution of batch jobs being 5 times as much as the interactive jobs.

2.7.2. *User Behavior: Sessions, Locality, Cycles, and Estimates*

2.7.2.1. Sessions

The idea of sessions was proposed first by [Zilber 2005] which demonstrates that CDF (cumulative distribution function) of think times, defined as the time between completion of a job and submission of next by the same user, has a steep climb at 20 minutes, thus assuming the jobs with 20 minutes or less think time between them to be in the same user session. In abstract terms, users tend to subsequently submit jobs in a session. [M.F.Arlitt 2000] proposes a similar idea, but in the context of web server logs. The idea is further pursued by [E.Shmueli, 2007] claiming that user behavior is more influenced by the response time than by slowdown, the former being the time from submission of job to its completion, and the latter being the response time divided by actual execution time. Reported CDFs for think time on several different workloads associate higher response times with lower percentages of jobs with a think time of 20 minutes or less, i.e. higher response time results in higher probability of user ending the session. In [E.Shmueli 2009], Shmueli et. al. make a similar conclusion by demonstrating a strong linear correlation between response time and think time. [D.G.Feitelson, 2008] also studies the relationship between users' think time and the response time, concluding that response time is a better predictor of users' reaction than slowdown.

2.7.2.2. Locality and Cycles of Activity

[D.G.Feitelson, 2008] demonstrates the locality of user behavior by presenting the difference between CDF of runtimes when taken across the whole log vs. across specific months or weeks, i.e. users tend to submit jobs with similar runtimes over smaller time scales. [E.Shmueli 2009] also claims temporal locality in the workload, i.e. users submit the same jobs over and over again, thus a similarity tendency by successive jobs of each user. Similarly, [D.G.Feitelson 2007] specifies more repetitiveness and regularity in the workload at smaller time slices, and references [R.Gibbons 1997], [D.Ferrari 1984] to claim workload data as non-stationary and changing as users learn to use a new system or as change the dominant application type, as opposed to the assumption made by workload generation methods. Two additional locality-related phenomena are specified as: 1) the humans daily cycle of work, and 2) autocorrelation of jobs, i.e. a correlation between runtimes of the same job, reversely proportional to the number of jobs separating the repetitions. A two-level workload generation method is proposed, where the top level picks the locality area to focus on, and the bottom level picks random jobs from that part of the distribution or population. [H.Li 2005] Argues based on [D.G.Feitelson 2002] that "workload traces are distributed with heavy tails and show a high level of self-similarity", thus runtimes are not similar across different time scales, probably leading to poor performance of global learning methods, e.g. neural networks. Authors thus use an instance-based learner to predict job runtimes of a month, trained on logs of the two preceding months.

In [E.Shmueli 2009], Shmueli et. al. simulate the behavior of users in submitting jobs by integrating 3 models: 1) session dynamics model, 2) job submission model, and 3) cycles of activity model. The first model incorporates the concept of sessions in the simulation of user behavior. The job submission model uses a two-level sampling process, with the top level generating the attributes for the jobs, and the bottom level repeating them to generate effects of locality. Repetition of job sizes, the base of the job submission model, is claimed using the corresponding CDF; however, the majority of job sizes are repeated only once: from 55% to 70% in all the traces. The last model divides trace data into day/night and weekday/weekend and uses the current day/time in the simulation to determine whether the model representing a user should be submitting jobs. ADAPS, proposed by [C.Glasnerlow 2011], is a prediction system which adapts to changes in the user behavior, via a) allowing or denying the use of sets of similar jobs (called clusters) in the runtime prediction, and b) assigning weights to sets of similar jobs and prediction methods when calculating the overall runtime prediction as a weighted average of predictions made by all possible pairs of predictor/similar-jobs.

In [E.Shmueli 2006], authors specify that the workload observed by the scheduler at any given time during the simulation is the combination of workload generated by all active user sessions. Each user session is composed of two parts: 1) a job submission behavior model, which specifies when the user submits more jobs and when he waits for jobs to complete, and 2) a work pool model that specifies the characteristics of the jobs. Authors claim that the users' job submission behavior is largely independent of the characteristics of the jobs that are submitted. Each session is associated with its own job submission and work pool model. The work pool model is composed of two distributions: the runtime model and the job size model. The work pools are modeled using empirical data drawn from trace data. Both job sizes and job runtimes are generated from the distributions of trace data. The distribution data on repetitions of jobs are also used: generated jobs are repeated according to this distribution.

2.7.2.3. Runtime Estimates by Users

Estimates of job runtimes provided by the users have been studied to uncover potential benefits, and to make possible their simulation. In [A.W.Mu'alem 2001], Mu'alem et. al. show that user runtime estimates are rather inaccurate. [Cirne 2001] claims that in four different traces, 50 to 60% of jobs used less than 20% of their requested runtime. Similar observations were made by [S.H.Chiang 2002]. [C.B.Lee 2004] specifies that users are quite confident of their estimations, and will likely not be able to provide better estimates. [C.B. Lee, 2006] studies whether the users can improve their estimate of runtime if there is reward for accuracy, and concludes that about half of the users do improve their estimates under these conditions, but there is not much improvement to the overall accuracy. The paper mentions the "padding hypothesis" as: users know their jobs' runtime, but pad their estimates to avoid the risk of jobs getting killed if they pass the estimation. To evaluate this hypothesis, the study asks users of a supercomputer center to provide non-kill estimates, with awards for accurate predictions; with result that seem to be the negation of the hypothesis as users still tend to overestimate. The study also conducted a survey to check whether users can provide a more expressive function of the importance of their jobs, or the utility function (user's satisfaction), and concludes that users are able to better express themselves.

In [D.Tsafrir 2005], Tsafrir et. al. build a model of parallel jobs and their associated user estimates through the study of several workload traces. The study shows all accuracy levels to be almost equally probable for the estimates, through demonstrating the flatness of histograms of the number of jobs vs. estimates' accuracy for successfully completed jobs across several workload traces (similar observations by [D.Tsafrir 2007]). Comparison of CDFs of actual and user-estimated runtimes further emphasizes this inaccuracy, which is only in the form of overestimation, as the job would otherwise be killed by the scheduler upon surpassing its estimated runtime. The study also reviews the existing models of user runtime estimates. The f-model introduces a "badness" f factor and assumes the users' runtime estimates fall between the actual runtime R and (f+1)R. The major flaw of the f-model is identified as the implicit provision of the relative order of jobs to the scheduler, i.e. short jobs are always reported shorter than long jobs, potentially improving the performance of backfilling. The ϕ-model, which generates estimates that result in histograms similar to those of actual estimates, is criticized for ignoring the cap that most platforms put on the runtime of a job, i.e. generating longer-than-cap estimates that never happen in practice. Existing models are also found to ignore the repetitiveness observed in the work of users of parallel machines, i.e. sessions, and the fact that user estimates compose a highly modal distribution: about 90% of the jobs in the examined traces use only 20 distinct values as user estimates. In a keynote speech [D.Tsafrir, 2010] Tsafrir adds to the above the ignoring of the use of the maximum allowed runtime as a favorite estimate. It can be concluded that not all existing models of user estimates can be used as components of a performance prediction method, particularly for its evaluation.

2.8. Challenges and Open Problems

Depending on the need the performance prediction aims to satisfy and the resulting situation, there are various challenges that need to be addressed. In this section, we describe a list of these challenges, and the set of techniques and approaches, if any, that are used to address them.

2.8.1. *Cross-Platform Performance Prediction*

A major challenge in performance prediction is the ability to make predictions for various platforms. This includes platforms which are not even available yet. The main motivation for cross-platform performance prediction is that it allows the scientists to decide which of the many available platforms to choose for running their application. It may also allow a cost-benefit analysis regarding the installation of a new platform or upgrading an existing one. The problem also relates to grid computing, as [F.Guim 2008] mentions that user estimates are only valid for homogeneous systems in a grid. Also, [K.Kurowski 2005] specifies as a major challenge the heterogeneity of systems on a grid and proposes modeling prediction errors to address these issues. The problems proposed by cross-platform performance prediction are: a) there may be no observations of the target application's behavior on the target platform, b) the target platform may be substantially different from the observed platforms, in terms of CPUs (speed, or even worse, architecture), the interconnect (bandwidth, latency, or even architecture), c) it may not be possible to obtain e.g. benchmark results or other dynamic-nature information for the target platform, due to e.g. not having access to it or it have not been built yet. This challenge can occur in many cases, examples are: acquiring a new machine, deciding where to run a particular set of applications on a grid, or making design decisions for an application or a platform. The methods proposed by the literature so far for dealing with the cross-platform performance prediction are described below. It should be noted that not all the methods listed here are designed to deal with the challenge; some of the methods partially achieve this goal as a beneficial side-effect of the innovative idea.

One category of approaches separately model the platform, rather than making it an implicit component of the application model, and thus are able to provide varying levels of accuracy and complexity in the platform model. However a mechanism needs to be provided for combining the two models, i.e. application and platform, in order to make performance predictions. Examples of the methods using this approach are [V.Taylor 2001], [L.Carrington 2003], and [G.R.Nudd 2000].

Another set of methods define a performance ratio between a base platform and the target platforms. The base machine is usually accessible easily, i.e. almost all observations have been obtained from it. The target machines on the other hand, are either not available at all, or are available for a limited set of observations, as in [J.Zhai 2010] which assumes the availability of one node of the not-yet-available new platform, or [M.Casas 2008] in which simulators are used on event traces of parallel applications obtained via instrumentation. These simulators are capable of simulating different interconnects and their corresponding parameters like bandwidth and latency [Dimemas 1997]. The performance, either actual or predicted, of the application on the target machine is then related to the performance on the base machine. The establishing of this relationship and the sophistication of the ratio itself varies greatly. In [W.Pfeiffer 2008] the proposed model provides the possibility of cross-platform runtime prediction (examined in the paper's experiments) as it formulates the coefficients partly on the basis of the ratio of the value of the predictor on a base and a target machine. [F.Nadeem 2006] translates performance of the application across grid sites using the assumption that the ratio of performance of the base problem size (the one executed on all sites) to that of any other problem size is constant across all grid sites. The performance translation is used for both creating a training dataset and for making predictions for platforms on which a specific problem size has not been executed. [J.Delgado 2010] Cross-platform prediction: specifies a "platform contribution" constant to model the CPU, which is identified via benchmarking. The resulting term is used as one of the factors that are multiplied to obtain the execution time. [M.Casas 2008] assumes that the ratio of application's average IPC (instructions per cycle) to the vendor-declared peak IPC, is uniform across all platforms (results suggest that this is a reasonable assumption if the two architectures are "close").

A final category of methods separate (or at least try to separate) the computation and communication of a target application, and model or simulate the target platforms' interconnects to obtain an estimation of the performance of the application on those platforms. The computation part is usually assumed to scale linearly depending on the base and target platforms. A major shortcoming of such methods is the assumption of a lack of overlap between computation and communication, which can have significant implications for the accuracy of predictions. [H.A.Sanjay 2008] addresses cross-platform performance modeling via scaling coefficients of different complexity functions which are components of the application model and are obtained on a reference platform as appropriate for a target platform, e.g. the computation complexity is scaled by a factor that is the ratio of applications runtime, for a "moderate" problem size, on reference vs. target platform. [L.Carrington 2003] falls under this category too; note that it also falls under the category of methods that separately model the application and the platform.

To summarize, although cross-platform performance prediction has been addressed extensively in white-box methods, very few black-box and gray-box methods have attempted this challenge, and none have actually addressed this challenge through a robust mechanism.

2.8.2. Problem Sizes and Input Parameters

The utilization of the increasing processing capability, available through clusters and grids, can be categorized into strong-scaling and weak-scaling. In strong scaling, the additional processing power is utilized to solve a larger instance of the same problem. In this category of usage, the total execution time of the application does not decrease significantly and may even increase, but the benefit is the solution of the target problem at a size which may not have been possible with fewer resources, e.g. due to insufficient memory per processing node. Weak-scaling, on the other hand, uses the additional processing power to solve the same problem size as with fewer resources, in a smaller amount of time; this reduction in the amount of processing time is the main benefit. Each of these categories of usage creates its own challenges for performance prediction. With strong-scaling, the main challenge is the prediction of the application's behavior, under a new platform, i.e. cross-platform performance prediction, and under a new the problem size. Weak-scaling faces the challenge of prediction under a different platform, as well as predicting the target application's scalability, since it may not be linear at all.

[D.J.Kerbyson 2005] uses expert knowledge of code and the problem it solves to model a scientific application's runtime as a function of different computation tasks based on the input parameters (problem size). [V.Taylor 2002] assumes that the different runs needed to generate all coupling values have the same input. The paper also explores how the coupling values change with a) the problem size and b) the number of processors. It is claimed that the changes with numbers of processors are finite, and correspond to different levels of memory hierarchy. Only the results corresponding to the length of kernel chains that produced the best predictions are shown. The kernels are not used to generalize to problem sizes and/or number of processors for which there are no data to calculate kernel coupling values, i.e. actual predictions. [J.Schopf 1998] and [A.Matsunaga 2010] assume detailed knowledge of applications' input parameters, e.g. in terms of knowing which ones have the most influence on the runtime. [X.Wu, 2004] examines the possibility of reusing kernel values, which are basically the mutual impact of different kernels that make up a parallel application, over different problem sizes and conclude that the kernel coupling values obtained for some problem classes (sizes) can be reused for others; more specifically, class B values can be used to predict performance for class A. This claim is made for the NAS benchmark suite, and SP benchmark is shown as a representative of SP, BT, and LU. However, this is a white-box method that assumes an understanding of the kernels of which a parallel application is composed. The method is also not general, i.e. works only when there is a limited set of problem sizes, not for various combinations of input parameters.

To summarize, the issue of input parameters and the resulting problem size seems to be requiring a lot further investigation, since the current literature does not seem to have answered several key questions, and also considering its high applicability to high-performance computing and grid computing.

2.9. Summary

In this chapter, we discussed performance prediction as a key research topic. We presented a detailed list of the areas in which performance prediction can provide important benefits. We provided a taxonomy of the state-of-the-art methods on performance prediction, and described in detail each category of existing research work. Next, we described a set of insights related to performance prediction, from both the research work that proposes novel prediction methods and from the research work that addresses an application area of performance prediction. Finally, we provided a list of challenges proposed by the application of performance prediction in different areas and under various constraints, and discussed the work done on each of these challenges and derived a list of open problems.

As a result of the survey presented in this chapter, we identified several open problems and gaps in the existing prediction methods. We next present these and subsequently describe how our performance prediction method addresses several key items of these challenges.

- Current methods are either expensive or not sufficiently accurate
- Current methods are not applicable in a production environment
- Many of the current methods depend on user/admin intervention
- Many of the current methods require too many input points
- Prediction across problem sizes is not addressed well by current methods
- Prediction across platforms is not addressed well by current methods

Based on the above survey, our understanding is that the following are the most important aspects of a prediction tool, which are not collectively addressed by any single prediction method: a) high prediction accuracy, b) requiring small number of input points, c) applicability in a production environment, and d) predicting across different problem sizes of a parallel application. Our Prediction tool, presented in the next chapter, addresses all these 4 challenges by implementing a prediction method that is: 1) highly accurate while requiring very few input points, 2) requires no user or OS-level support and is computationally feasible to run in a real world scheduling environment, and 3) is capable of predicting runtime and speedup for different problem sizes of a parallel application.

CHAPTER 3[1]

ADEPT Runtime and Speedup Prediction

3.1. ADEPT's Goals

Adaptive CPU resource allocation is a widely researched topic in job and grid scheduling with potential to improve response times significantly (up to 70%) by reducing fragmentation and considering the current machine load [V.K.Naik 1997][W.Cirne 2003][A.C.Sodan 2006][L.Barsanti 2006]. Due to typical efficiency curves, the latter contributes most to the benefits and means running applications with more resources if the load is light and with less if the load is heavy [V.K.Naik 1997][A.C.Sodan 2009]. Adaptive resource allocation is a practically promising approach, considering that a study found that 98% of the users said their applications could adjust to different resource allocation at start-time [W.Cirne 2003]. Adaptive resource allocation depends on efficiency curves per problem size (strong scaling) since efficiency-based allocation was found superior to uninformed approaches like equal resource partitioning [S.H.Chiang 1996]. However, efficiency/scalability curves are not generally available; this is a major reason why adaptive resource allocation is not yet incorporated in practical schedulers. Thus, providing scalability prediction in an easy-to-use manner would open new possibilities for better practical scheduling. Users may also select job sizes "tactically" under considerations of trading shorter waiting times for increased runtimes. Scalability prediction is also relevant for determining the maximum meaningful CPU resource allocation to a parallel job (and therefore an often-tackled problem, e.g. [X.H.Sun 1999]) as feedback to users and system administrators. Though so far mostly applied on clusters, with the emergence of parallel computing in every-day

[1] *This chapter incorporates the outcome of a joint research undertaken in collaboration with Jacob Machina under the supervision of Dr. Angela Sodan. See the declaration of co-authorship for details.*

life on multi-core systems, adaptive schedulers will likely increase in practical relevance. This is especially true if the resources allocated to a virtual-machine running parallel jobs can vary [A.C.Sodan 2009]. Luckily, OpenMP applications on multi-core SMP servers were found to exhibit similar shapes of speedup/runtime curves as MPI applications on clusters [M.Curtis-Maury 2005]. This opens the possibility of applying the same scalability prediction approach.

Accurate predictions can be obtained via either black-box or white-box approaches. The latter are based on application-internal and machine information, require code instrumentation, compiler/OS support, analysis of memory-access behavior, simulation, etc. [L.Carrington 2003][B.Lafreniere 2005][G.Marin 2004] [X.H.Sun 1999]. Thus, white-box approaches are complex and computationally expensive, making them unsuitable for large-scale use in supercomputing centers though indispensable for cross-site prediction or projection of performance on not yet practically available platforms. Black-box approaches predict scalability (speedup and runtime) using only runtime observations on different numbers of nodes, by assuming conformity to a simple descriptive model which can be fitted to the observations to derive a specific model instance. The required observations can easily be obtained from data routinely collected in historical databases by supercomputer centers or from explicit test series. This makes black-box approaches much easier and much cheaper to apply, though, to be practical, the number of required observations needs to be small. Currently existing black-box models suffer from applications potentially deviating significantly from the models because of anomalies or because exhibiting specific scalability patterns which cannot be directly explained by the model.

Our overall goal is scalability prediction (in the sense of strong scaling), on both multi-core SMP servers and clusters, which is practically feasible for production environments. To enable production use, we apply a black-box approach based on the Downey model shown to capture simplified behavior of parallel applications very well

[A.Downey 1997 Model]. The Downey model has been around for a long time but has not been widely used due to many real applications not fully conforming to the model, e.g. by showing super linear speedups, and due to reliability of a specific prediction being hard to judge.

As described in [A.Deshmeh 2010], with the development of ADEPT (Automatic Downey-based Envelope-constrained Prediction Tool), we pursued the following detailed goals:

- Achieve high prediction accuracy, while requiring only few observations (typically 3 to 4).
- Provide a computationally efficient approach for deriving the model instance.
- Identify cases where the application does not fully conform to the Downey model as anomalies, with automatic correction and multi-phase modeling for individual irregular points and typical patterns.
- Perform reliability judgment which recognizes unsuitable observation layout and proposes placement ranges of additional observations.

To address these problems, ADEPT employs a special envelope-derivation technique which constrains the search for the best-fitting model instance, a special metric for detection of anomalies, and special pattern handling for cases like super-linear speedup. Experiments with the NAS benchmarks [D.H.Bailey 1995] and seven real applications show the efficiency and prediction quality of ADEPT in handling normal cases and anomalies. We obtained generally above 80% prediction accuracy, even in cases with anomalies and for predictions which extrapolate for more than twice the number of nodes that were used in the closest observation. The experiments also demonstrate the effectiveness of reliability judgment.

3.2. Related Work

We next provide a brief description of the literature most significantly related to ADEPT. Black-box approaches attempt to provide accurate predictions with low overhead by assuming conformity of parallel applications to an underlying model to which available data is fit. The approach in [R.Gibbons 1997] uses historical information of a parallel application, including number of nodes and user estimate, as input to a weighted least squares method for obtaining a quadratic runtime formula, which can then be used to make predictions. The method proposed in [W.Smith 2004] also employs historical information, but obtains the predictions from a job's corresponding "group of similar jobs", using linear regression, or in some cases averaging. Groups of similar jobs are determined using greedy and genetic algorithm search. The technique proposed in [B.Lafreniere 2005] applies multiple linear regressions to historical information to extract the value of parameters of the rough, user-provided complexity formula. This quantizes the rough formula, which can be used to make predictions. Downey et al. propose a black-box model which uses only two parameters, called average parallelism and variance of parallelism [A.Downey 1997 Model]. To validate the proposed model, the NAS benchmark suite [D.H.Bailey 1995] was used to generate runtime data for model fitting. However, all observations were used to train the model; no predictions were made. Black-box approaches benefit from zero overhead for the target application at runtime and no need to access the source or binaries, but are faced with the challenge of determining the optimum model instance. An adaptive runtime method for determining the maximum number of tasks meaningful for execution by OpenMP [OpenMP 2008] threads is proposed in [A.Duran 2008]. The approach measures work per task and overhead to decide whether tasks should be created at a certain nesting level but does not provide any predictive model.

Most white-box methods adopt one of two approaches: perform independent code and machine profiling then combine these to produce predictions, or use code-instrumentation on a specific code-machine combination to construct a model of application behavior. The approach proposed in [G.Marin 2004] extracts a target application's key performance characteristics from its binary. This approach constructs models of memory access behavior and maps them on the target architecture to provide runtime predictions. The approach proposed in [A.Snavely 2001] also employs independent modeling of the application (memory access and communication behavior) and the target architecture (capability to perform load and store operations), and maps the former on the latter to provide predictions. Closely related is the technique described in [L.Carrington 2003], which models both the application and the architecture based on their "fundamental operations" capability. The SCALA system [X.H.Sun 1999] uses the concept of scalability of code-machine combinations to make inter-platform predictions, and reduces the time complexity of the modeling by determining key basic blocks. Another approach is proposed in [B.Barnes 2008], which employs regression to predict scalability. As indicated by [B.Barnes 2010], the capability to address different problem sizes when predicting runtime and speedup of parallel applications is highly beneficial to adaptive resource allocation, but is currently only addressed by white-box tools and not feasible in production environments.

Gray-box methods aim for the best of both previous categories, i.e. high accuracy of white-box and low overhead of black-box methods. The term was introduced by [B.Barnes 2010], even though older examples of the approach can be found in the literature: [E.Ipek 2005], [B.Lafreniere 2005]. In [B.Barnes 2010], Barnes et. al. propose a method that uses similarity in the parameter space to predict parameter values that would result in time-constrained scaling, i.e. increasing problem size to maintain constant performance on increasing numbers of processors. In [B.Lafreniere 2005], Lafreniere et. al. propose a method that depends on user-specified "rough" linear formula to relate performance to application's characteristics and input parameters. Model's coefficients are determines using regression over a dataset of performance versus independent variables. In [Nirav 1999], authors relate the runtime to the input parameters using K nearest neighbors, K nearest neighbors with weighted averaging (weights are the reverse of distance of the neighbor from the target point), and locally weighted polynomial regression. To summarize, the existing work in this category is still not applicable in a production environment due to its requirement of internal knowledge on the target application and/or user intervention.

3.3. The Downey Model

3.3.1. *Overview*

Downey proposed a black-box model which describes an application via two parameters: A as the average parallelism and σ which is the variance in parallelism, i.e. describes the shape of the curve [A.Downey 1997 Model]. The model thus has a semantic meaning related to typical application behavior. It provides piecewise functions for the application's speedup and runtime, specified separately for low variance and high variance modes of the model. In Table 1, n represents the number of nodes, T(n) and S(n) represent the runtime and speedup on n nodes. To conform to Downey model, which states that $T(\infty)=1$, we assume all runtime values are divided by this value. Figure 2(a) and (b) show a set of speedup curves constructed using the Downey model with different A and σ values. A smaller σ means the parallel application reaches its maximum speedup at a smaller number of nodes. σ=0 corresponds to linear speedup.

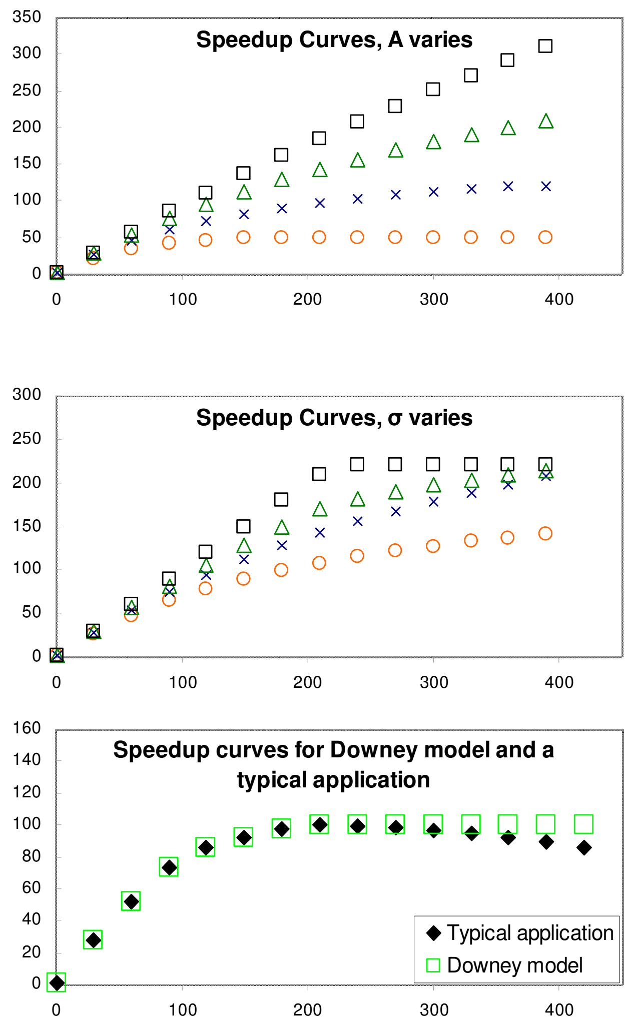


Figure 2. Downey model speedup curves
(top) Speedup curve: σ=2, varying A (1000, 300, 120, 50), (middle) Speedup curve: A=220, varying σ (0, 0.5, 1, 1000), (bottom) Downey model's lack of support for declining piece of the speedup curve. Graphs show S over N.

3.3.2. *Strengths and Weaknesses*

The Downey model benefits mainly from the fact that it uses only two parameters (namely, A and *σ*). This makes the model easier to store and understand, and reduces the number of observations necessary to learn the parameters for a

Table 1. S and T piecewise functions of Downey model.

Mode	*n* range	*S(n)*	*T(n)*
Low Variance	$1 \le n \le A$	$\frac{An}{A+(\sigma/2)(n-1)}$	$\frac{A-\sigma/2}{n}+\sigma/2$
	$A \le n \le 2A-1$	$\frac{An}{\sigma(A-1/2)+n(1-\sigma/2)}$	$\sigma(A-1/2)/n$ $+1-\sigma/2$
	$2A-1 \le n$	A	1
High Variance	$1 \le n \le A+A\sigma-\sigma$	$\frac{nA(\sigma+1)}{\sigma(n+A-1)+A}$	$\sigma+\frac{A+A\sigma-\sigma}{n}$
	$A+A\sigma-\sigma \le n$	A	$\sigma+1$

specific application, i.e. to construct an application's corresponding Downey model instance.

A typical speedup curve has 4 pieces: approximately linear, transitional, flat, and declining. However, the Downey model does not include parallelism overheads such as communication cost, and therefore does not capture the declining section, the main drawback of the Downey model; see Figure 2(c). This is insignificant as the maximum meaningful number of nodes can be obtained as $2\mathrm{A}-1$ for low variance mode and as $\mathrm{A}+\mathrm{A}\sigma - \sigma$ for high variance mode, i.e. there is no need to allocate more cores to an application than these maximum values, and hence the behavior of the model beyond these maximums can be disregarded without loss of generality. Also, the processor working set—proposed as a metric to determine a balance between speedup and resource consumption [D.Ghosal 1991]—could be calculated using the fitted model, by finding the minimum *n* such that $\eta(n)$ is maximal, with $\eta(n) = S^2(n)/n$.

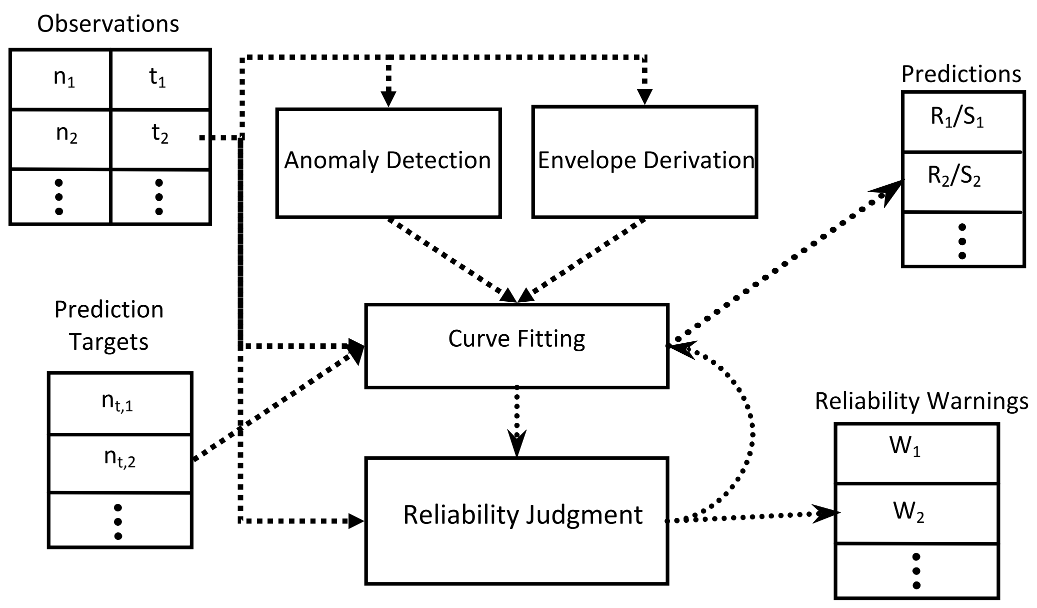


Figure 3. ADEPT components; Arrows show information flow

3.4. The ADEPT Predictor

ADEPT uses an instance of Downey model to make its predictions. Therefore, to obtain this instance, ADEPT needs to learn A and σ from a set of observations, each being a specific number of nodes paired with its corresponding runtime. Note that the serial runtime *T(1)* may not be available which makes predictions more difficult as the actual speedup values cannot be determined. Moreover, real applications may significantly deviate from the Downey model, either in terms of an individual anomalous point or of a specific scalability pattern which the model does not natively incorporate. Even for applications that closely conform to the model, input points may all be drawn from the linear section of the scalability curve, or be placed such that vastly different model instances still explain them. The latter happens when there exist several Downey model instances that happen to fit the observations equally well, while having substantially different values for the parameter A, due to the effects of the parameter σ. The Section on reliability judgment provides the details on how this is detected and handled. To address these challenges and provide an efficient predictor which is applicable in production environments, ADEPT is composed of four major components (see Figure 3):

1. Anomaly detection, which identifies individual anomalous points and specific scalability patterns typical in some HPC applications.
2. Envelope derivation, which significantly constrains the search space.
3. Curve fitting, which finds a model instance within the envelope for each prediction target.
4. Reliability judgment, which performs post-processing to detect unreliable predictions.

Envelope derivation and curve fitting constitute the core of the ADEPT tool and derive the predictive model. Envelope derivation reduces the search space of model instances to those which could explain observations, making fine-grained search feasible. Anomaly detection and reliability judgment enhance ADEPT with features necessary to handle real applications. The algorithm used by ADEPT is as follows. The more detailed algorithm corresponding to each step is presented in the corresponding section.

```
1. Obtain the envelope, E, from I, the set of observations:
   E = EnvelopeDerivation(I)
2. Obtain the list of Adjusted Weights, W, from I:
   W=AnomalyDetection(I)
3. Obtain the set of predictions, P, which is one prediction for each of the
   targets in the set T:
   P=CurveFitting(I, T, E, W)
```

4. Generate Reliability Warnings for the set of Predictions: WR=ReliabilityWarning(I, T, P)
5. If more input is both required and available, add new input to the input set I, and Go to Step 1.

First, we will discuss the core of the ADEPT predictor, and then show experiments which demonstrate its effectiveness for normal cases. We will later present ADEPT's anomaly handling and reliability judgment and corresponding experiments.

3.5. *Obtaining the Predictive Model with ADEPT*

3.5.1. *Envelope: Deriving Constraints from Observations*

As mentioned before, the goal of the envelope derivation step is to make exhaustive search feasible via reducing the search space. This goal is achieved by establishing an envelope, which is a set of constraints on the parameters of the model.

The envelope is created using the following idea. For observations which perfectly match a model instance, a closed-form solution could calculate exact parameter values. For real applications which do not match perfectly, we assume each input point deviates from the underlying model by at most δ up or down. Then measured runtimes can be mapped into a range in which the runtime predicted by the underlying model must fall. These ranges can be used for pair wise calculation of closed-form solutions for the lower and upper constraints.

Formally, we assume the existence of a model instance m_i for a real application app_i, with a maximum deviation δ (a fraction) from m_i at any observation point: $T_{m_i}(n) \in \left[T_{app_i}(n)*(1-\delta), T_{app_i}(n)*(1+\delta)\right]$. Since m_i is not known, δ must be guessed. To test the validity of this guess, runtime values provided by any model instance assumed as m_i can be compared to actual observations as: $\delta \geq \max\limits_{observations}\left(\left| T_{actual} - T_{predicted} \right| / T_{actual}\right)$. If this test fails, our initial guess for δ was incorrect, and δ can be incremented until it passes.

The envelope is defined as a set of range pairs whose first and second components specify constraints on A and σ values, respectively: $E = \left\{ c_i = \left(\left[A_{i,\min}, A_{i,\max}\right], \left[\sigma_{i,\min}, \sigma_{i,\max}\right] \right) \middle| \, i = 1, \ldots, K \right\}$. Each range pair thus represents a lower-bound model instance: $\left(A_{i,\min}, \sigma_{i,\max}\right)$, and an upper-bound model instance: $\left(A_{i,\max}, \sigma_{i,\min}\right)$ as constraints (see Figure 4). The envelope consists of all model

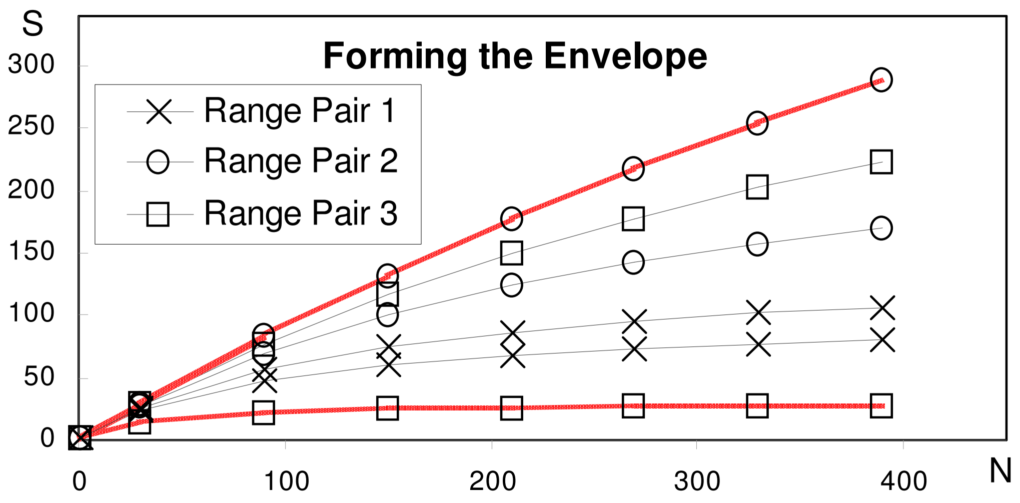


Figure 4. Forming the envelope
Range Pair 1 is redundant and discarded. Range Pairs 2 and 3 are combined to form the envelope, with absolute bounds shown via heavier lines.

instances lying between the bounds. In the experiments, we only show lowest and highest bounds of all pairs. The envelope is not to be confused with a confidence interval; it only constrains the set of model instances.

We have derived formulas and their extensions using δ for all possible pieces of the low and high variance modes of the Downey model. The complete set of ten formulas and their derivation are described in [A.Deshmeh 2009] and in Appendix A. The following equations give examples of closed-form formulas for the first piece of the low variance mode of the Downey model, and the corresponding formulas with the δ parameter included:

$$\sigma = \frac{2\left(n_j t_j - n_i t_i\right)}{n_j - n_i} \tag{1}$$

$$A = n_i t_i - \sigma \frac{n_i - 1}{2} \tag{2}$$

$$\sigma \in \frac{2(n_j t_j - n_i t_i)}{n_j - n_i}[1-\delta, 1+\delta] \tag{3}$$

$$A \in [n_i t_i(1-\delta) - \sigma(n_i - 1)/2,\ \ n_i t_i(1+\delta) - \sigma(n_i - 1)/2] \tag{4}$$

These formulas assume that both observations lie in the same piece. For any three observations, at least one pair satisfies this assumption and holds the final model instance, while other range pairs merely increase the search space.

Therefore, the algorithm for the envelope derivation step uses the following actions to establish the envelope:

- Use the set of observations, $I = \{ (n_i, t_i) \mid i = 1,\dots,M \}$, to form all pairs of observations, $P = \left\{ \left((n_i, t_i), (n_j, t_j) \right) \middle| \forall i, j \in \{1, \dots, M\}, i < j \right\}$, where M is the number of observations.

- For each observation pair $p \in P$, calculate all possible range pairs for the parameters, $C_p = \{ ([A_{i,\min}, A_{i,\max}], [\sigma_{i,\min}, \sigma_{i,\max}]) \mid i = 1,\dots,k_p \}$, where k_p is the number of possible range pairs based on the pair of observation and the number of formulas we have derived to calculate range pairs, as detailed by Appendix A. For each observation pair *p*, a maximum of four range pairs are possible, two for each of the low and high variance modes. This gives the set $C' = \bigcup C_p$.

- Discard redundant range pairs in C'. A range pair $c_l = (\lfloor A_{l,\min}, A_{l,\max} \rfloor, \lfloor \sigma_{l,\min}, \sigma_{l,\max} \rfloor)$ is redundant if there exists some $c_j = (\lfloor A_{j,\min}, A_{j,\max} \rfloor, \lfloor \sigma_{j,\min}, \sigma_{j,\max} \rfloor)$, such that the following conditions hold: $\lfloor A_{l,\min}, A_{l,\max} \rfloor \subseteq \lfloor A_{j,\min}, A_{j,\max} \rfloor$ and $\lfloor \sigma_{l,\min}, \sigma_{l,\max} \rfloor \subseteq \lfloor \sigma_{j,\min}, \sigma_{j,\max} \rfloor$. In other words, all the model instances that fit into constraints of c_l also fit into constraints specified by c_j but the reverse does not necessarily hold. This means discarding c_l while keeping c_j would not modify the set of model instances that are examined, as shown in Figure 4. The result of discarding those range pairs which are redundant is the final set of range pairs $E = \{ c_i \mid i = 1,\dots,k,\ k \le |C'| \}$.

Table 2. Comparison of runtime of ADEPT with three curve fitting methods
Methods compared are exhaustive search, genetic algorithm, and Levenberg-Marquardt. Comparison is made for two sets of curve fitting experiments, one shown in each row. Runtimes shown in each row are averages over the experiments in that set.

Experiment Attributes		**Runtime (sec)**			
A range	*σ* range	Levenberg-Marquardt	Exhaustive Search	Genetic Algorithm	ADEPT
400 to 1000	0.0 to 1.0	0.08	5	14	0.46
400 to 2000	1.1 to 12.0	0.07	5	14	0.48

3.5.2. *Curve Fitting: The Search for an Optimal Model Instance*

The curve fitting step finds an optimal Downey model instance for each prediction target. Rather than generating a single model instance, we can find one which is specifically biased towards a single prediction target. This is accomplished by assigning weight according to the relevance. For extrapolative speedup prediction, the closest observation typically best shows the trend.

The input to curve fitting are the observation points, $I = \{ (n_i, t_i) \mid i = 1, \ldots, M \}$, the envelope to which the search is limited, $E = \{ c_i \mid i = 1, \ldots, k \}$, and the number of nodes on which a prediction is needed, n_{target} . Multiple inputs per job size are handled by dropping obvious outliers and otherwise averaging inputs to avoid an overly high weight for the repeated job size. The output of curve fitting is the best fitting model instance found: $(A_{final}, \sigma_{final}) = CurveFitting(I, E, n_{target})$.

Our optimality criterion for curve fitting is the Weighted Sum of Squared Relative Errors (WSSRE). The weight of a point is calculated as:

$W_i = q * \max\left\{ \left| n_{target} - n_j \right| \;\middle|\; j = 1,\ldots,M \right\} - \left| n_{target} - n_i \right|$, where the factor *q* determines how sensitive the weights will be to prediction distance, with smaller values being more sensitive. The value of q can be selected by cross-validation, i.e. the value that results in the highest prediction accuracies is selected. In our experiments over different applications, we found that a value of 2 for this parameter resulted in the highest prediction accuracies.

The exhaustive search is performed in two passes to further reduce the search space. The first pass is a one-dimensional search, and the second pass is a local two-dimensional search. The one-dimensional search constrains the search space to only those model instances which pass directly through the input point closest to the prediction target, which we call the fixed point. ”Fixing” this point, i.e. only considering model instances that generate this point, allows us to calculate values for *σ* from any value of *A corresponding to the model instances that fit the fixed point*. Please note that we perform exhaustive search, and thus its first step, only on those values of *A* which fall within the envelope. The one-dimensional search finds a model instance that fits the observations well, in linear time. We then find the best fitting model instance byperforming two-dimensional exhaustive search, varying the *A* and *σ* values, obtained at the first pass, with fine-grain steps up to 15% within the envelope, to obtain the final model instance.

3.6. *Effectiveness of ADEPT’s Curve Fitting*

We have conducted experiments to demonstrate the superiority of the combination of curve fitting and envelope derivation components of ADEPT, over other curve fitting. We compared ADEPT with three methods: exhaustive search, genetic algorithms, and the Levenberg-Marquardt method [K.Levenberg 1944], a common optimization approach. We used its implementation levmar [M.I.A.Lourakis 2005], with default settings, and arbitrary initial guesses for the parameters. The genetic algorithm implementation used is GALib [M.Wall 2009]. Boundaries within which to search were set as A: 1 to 3,000, and σ: 0 to 3,000 for all three methods; 10 observations were generated from the Downey model (perfect match is possible), 4 of which were provided as input. Two sets of experiments were run to cover cases of low and high variance, with each set covering four different experiments.

Figure 5 shows a representative prediction example (from the second set). ADEPT, the genetic algorithm, and the exhaustive search all made perfect predictions which hence overlap. The Levenberg-Marquardt method, however, made highly inaccurate predictions. We found the method to be highly sensitive to the initial guesses of A and σ for up to 2000 iterations.

To compare the cost of running each of the methods, average runtimes are shown in Table 2. The Levenberg-Marquardt method and ADEPT were both very fast,

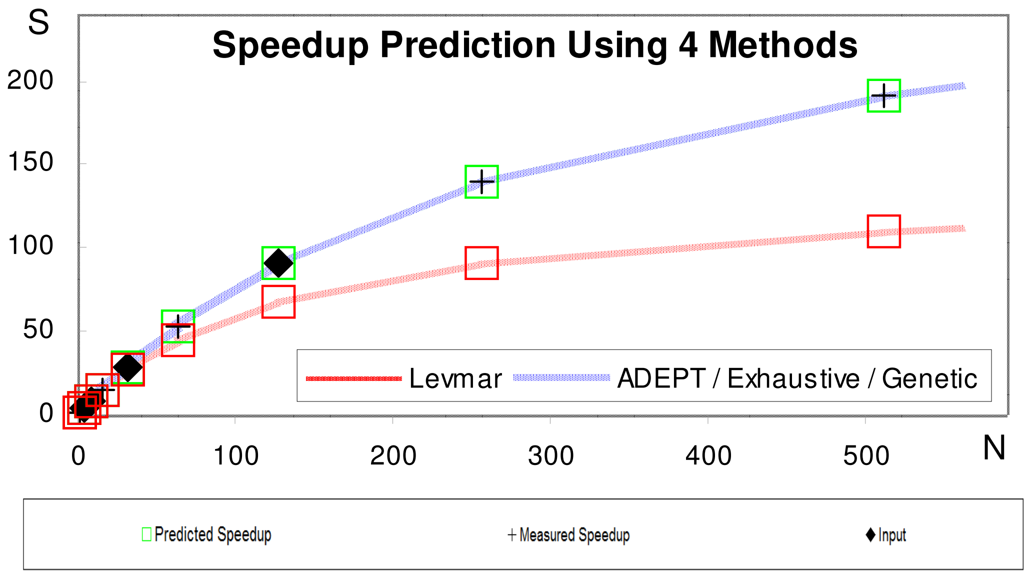


Figure 5. Speedup prediction of ADEPT, GA, exhaustive search, and Levenberg-Marquardt. The first three made perfect predictions (higher trend line), while the fourth was inaccurate (lower trend line).

with less than 100 ms and less than 500 ms runtimes, respectively. Exhaustive search and genetic algorithm had average runtimes of 5 sec and 14 sec, i.e. were 10 and 30 times slower than ADEPT.

The presented experiments demonstrate that ADEPT combines the high accuracy of exhaustive search and genetic algorithms and the speed of the Levenberg-Marquardt method.

3.7. Experimental Setup

To validate the power of ADEPT, we use two groups of applications. The first group includes MPI and OpenMP implementations of BT, CG, FT, LU, and SP from the NAS benchmark suite, Class B [D.H.Bailey 1995]. We ran these benchmarks on clusters of SHARCNET [SHARCNET 2009], with three runs per benchmark and per number of nodes. OpenMP benchmarks were run with four threads per CPU on a node with 8 quad-core CPUs. NAS class B was used because we needed scalability curves with transitional (nonlinear) phases and we had only up to 256 cluster nodes available.

The second group consists of seven real applications, which also were run in the same environment. The data originates from scalability tests, performed by system administrators to approve major resource requests by intensive users. The applications themselves were, however, kept anonymous, and we therefore call them App_A to App_G. However, it is known that users cover a broad range of domains such as physics, chemistry, and economics.

As mentioned before, *T(1)* is key information which is not generally known for real parallel applications. This is the case for four of the real test applications, and we use estimated *T(1)* to draw speedup curves. When *T(1)* is available, we omit *T(1)* for some tests.

The tests include predictions for both interpolation (targets falling between at least two input observations) and extrapolation (targets not between any two input observations). The evaluation criteria used throughout the experiments are relative error percentage, *E*, and prediction accuracy percentage, *PA*, defined as $E = \left(|predicted - actual| / actual \right) * 100$ and $PA = 100 - E$, respectively.

For the one-dimensional phase of the curve fitting step, increments for *A* were determined by dividing the envelope into 5,000 evenly distributed values. For the two-dimensional phase, *A* and *σ* assumed 500 evenly distributed values each, evaluating 250,000 instances of the model. More fine-grained search did not generally increase prediction accuracies; the chosen search granularity was seen as a balance between speed and accuracy. For most tests, *q* was set to 2. The value of this parameter was selected using cross-validation, i.e. a value of 2 resulted in the highest prediction accuracy in our experiments.

3.8. *Experimental Results for Model Derivation*

3.8.1. *Speedup Prediction*

We first demonstrate the performance of ADEPT in speedup prediction for normal cases. The results are shown in Figure 6 and Figure 7. We show predictions, measured values, and input points. The number of runtime measurements used as input is either 3 or 4 for all the experiments. For some applications, two graphics are shown; the first one does not include *T(1)* in the input, and the second does. *T(1)* is indeed unavailable for App_B, App_D, App_E, and App_F.

The results show generally very good accuracies, with the exception of 27% error for CG at 2 nodes. Inclusion of *T(1)* as input did not result in significant improvement in prediction accuracy. Since the applications App_C, NAS_CG, and NAS_LU do not include *T(1)*, the envelope (calculated on runtime) cannot show speedup properly. For NAS_FT, two curves are shown, one with uniform weighting of points, and one using a *q* value of 1.01 which shows better extrapolative prediction, validating ADEPT's biased curve-fitting approach.

Comparing accuracies of interpolations vs. extrapolations, BT extrapolations (2%, 8%, 1%, 4% errors at 144, 169, 196, 225 nodes) were comparable to interpolations (5%, 1%, 0%, 3%, 1% at 36, 49, 64, 81, 100 nodes). ADEPT showed more accurate interpolations (2% error at 8 nodes) than extrapolations (22% and 13% errors at 64 and 128 nodes) for CG without *T(1)*. FT had a 23% error for extrapolation at 128 nodes, and 23% error for interpolation at 16 nodes, though its other interpolation errors were below 9%. The experiments for speedup prediction thus do not conclude generally higher accuracies for either interpolation or extrapolation.

Regarding the distance of extrapolation, i.e. how far ADEPT can predict, errors of 12% and 9% were measured at 196 and 225 nodes for BT, when a maximum of only 81 nodes were used as input. For CG, using a maximum of 32 nodes as input resulted in a 20% error at 128 nodes, while using a maximum of 64 nodes results in 9%, 11%, and 14% errors over three experiments. FT shows an error of 47% at 128 nodes when a maximum of 32 nodes is used as input, and an error of 23% when using a maximum of 64 nodes as input. The speedup curve of FT shows that for a maximum of 32 nodes as input, predicting the actual speedup value at 128 nodes is simply not possible using any black-box method. Highly accurate extrapolations were observed on generally more than twice the maximum number of nodes used as input.

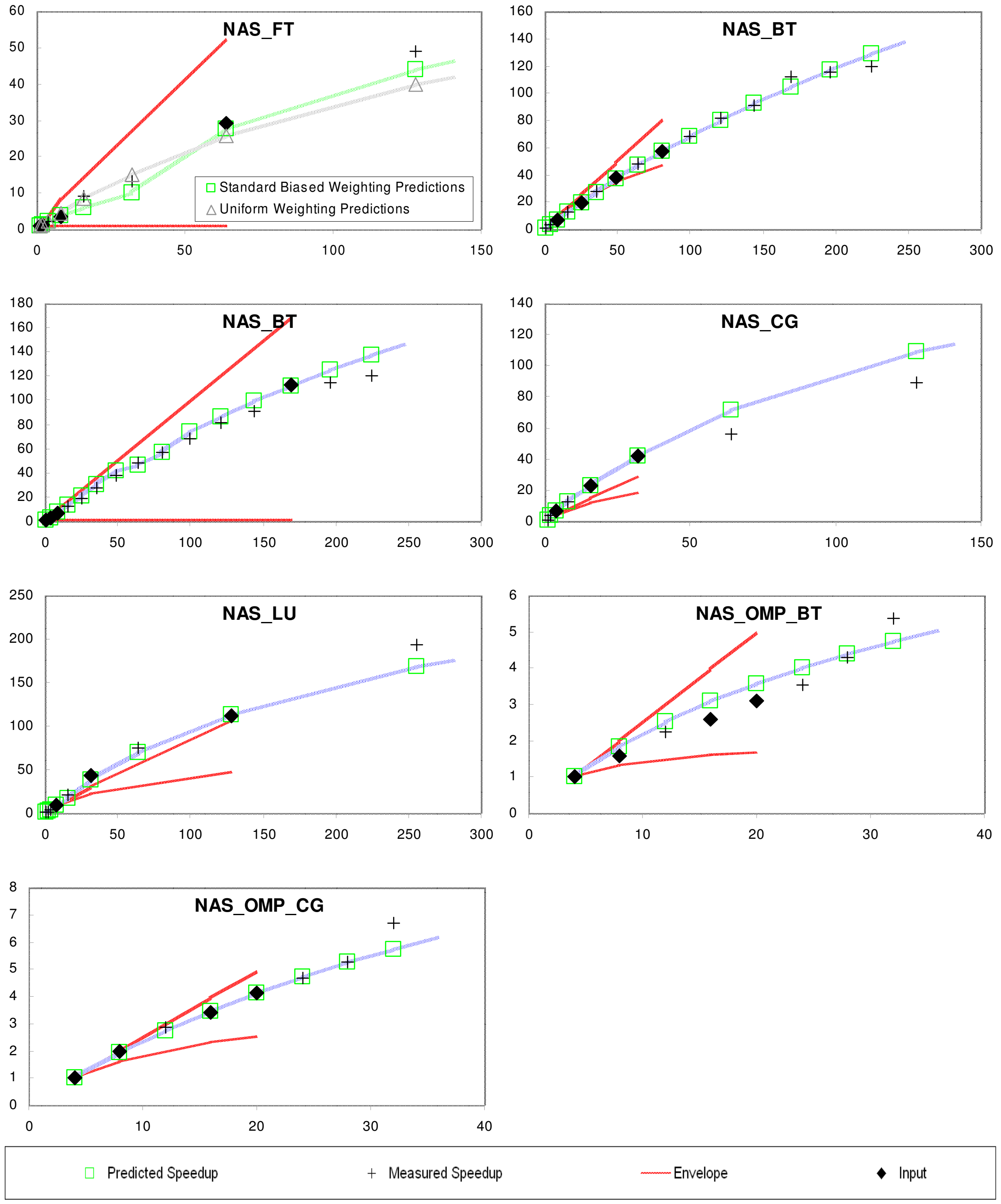


Figure 6. Speedup prediction results for NAS benchmarks
Results are for MPI implementation of NAS BT, CG, FT, and LU, and the OMP implementation of NAS BT and CG, both interpolation and extrapolation. Graphs show S over number of threads for OMP benchmarks, S over N otherwise.

Additional experiments investigated placement and using more (up to 6)

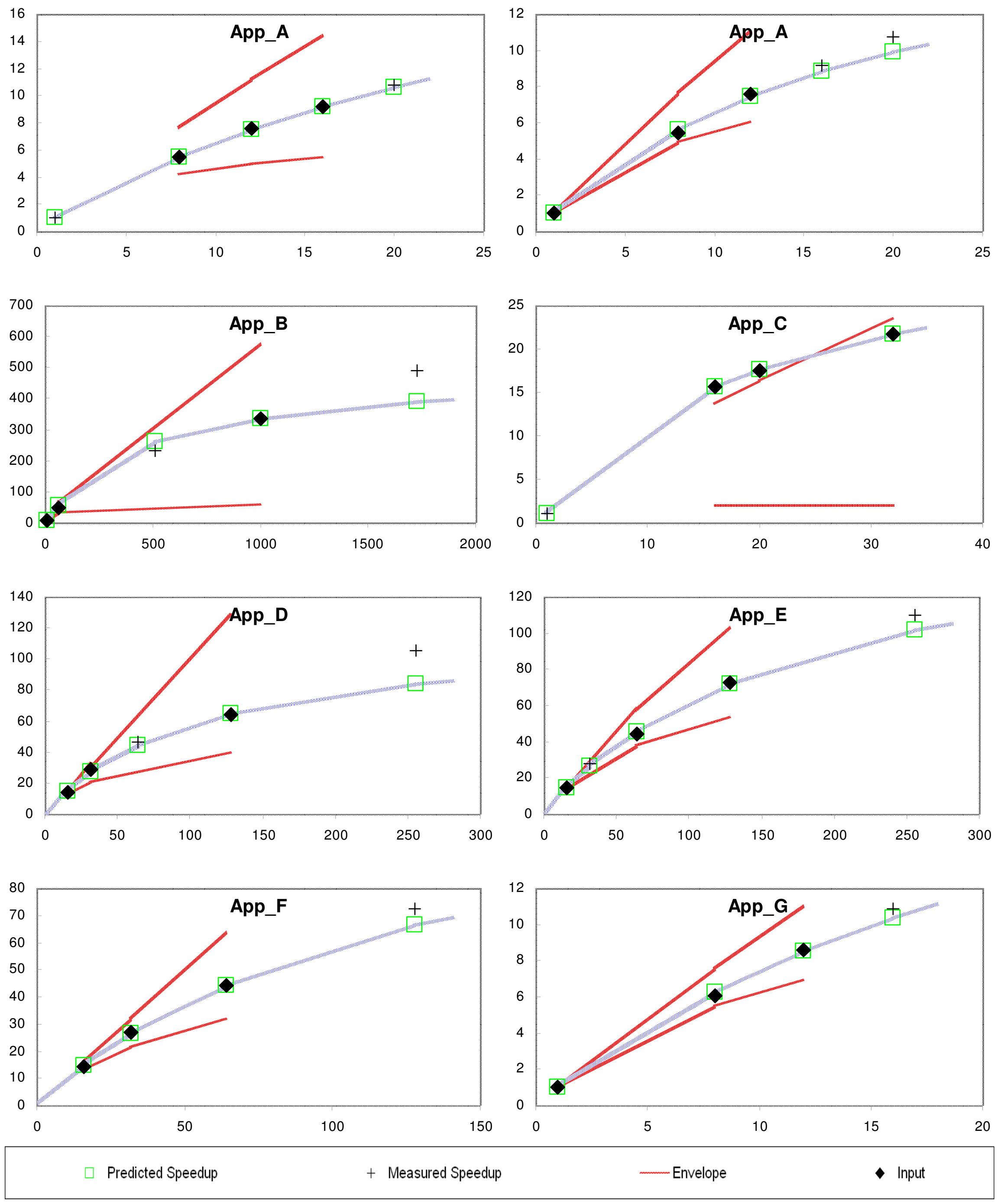


Figure 7. Speedup prediction results for the anonymous real world applications Both interpolation and extrapolation. Graphs show S over number of threads for OMP benchmarks, S over N otherwise.

observations. Regardless of the layout of input points, generally very high prediction accuracy was obtained as exemplified by NAS_BT with *T(1)*. This holds as long as not all the points are in the linear section of the speedup curve (detectable by reliability judgment, discussed later in Section 3.10). More input points also did not generally result in any accuracy improvement.

3.8.2. *Runtime Prediction*

Next, we demonstrate the performance of ADEPT at runtime prediction. Results are shown in Figure 8. Results for App_A, App_C, and App_G are not shown. This is because, the speedup prediction accuracies for these benchmarks were generally above 90%; the same applies to all runtime predictions for these applications, for both interpolation and extrapolation. The runtime predictions for these three applications are so accurate that measurements and predictions would overlap on the runtime curve, and hence we omit them from result presentation. The number of runtime measurements used as input was either three or four for all experiments. The runtime axis has logarithmic scale to better separate the points. We show a single graphic per application or benchmark, which corresponds to an experiment that does not use *T(1)* as input, since the provision of *T(1)* as input does not result in any major improvement in the runtime prediction accuracy. Accuracies obtained for runtime prediction, excluding *T(1)*, were generally above 80%, with the exception being CG with 36% error at 2 nodes, FT with 36%, 34%, 45%, 28%, and 29% prediction errors at 2, 4, 8, 32, and 128 nodes.

Regarding the accuracies of interpolations vs. extrapolations, and influence of extremity of extrapolation, the same trend applies to runtime prediction accuracies as discussed for speedup predictions. We also performed tests to examine the effect of increasing input size from three points to four points on accuracy of runtime prediction. Results do not show any significant improvement in accuracy of predictions for any benchmark or real application. The only trend proven, as a byproduct of these

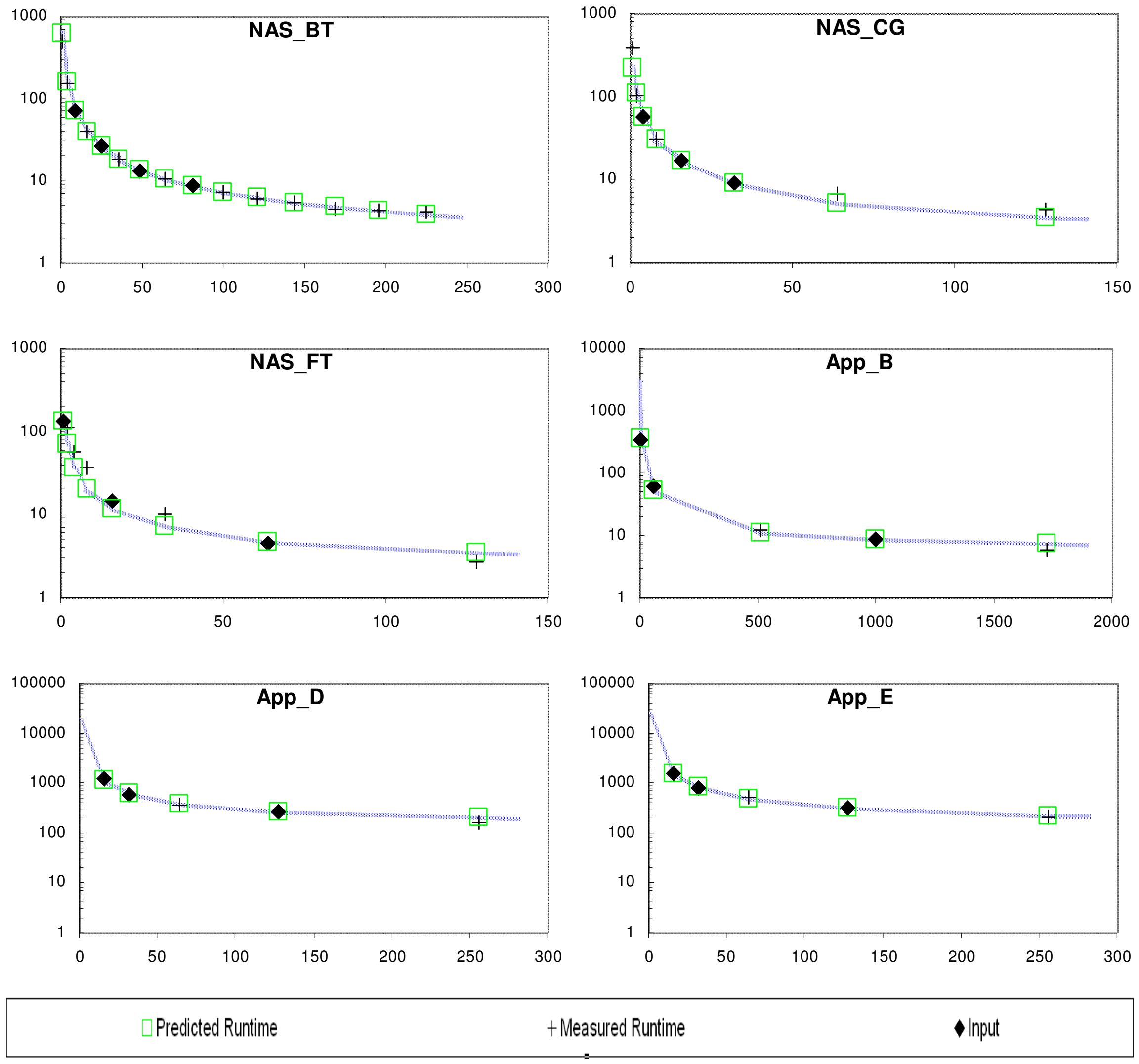


Figure 8. Runtime prediction results for NAS benchmarks and anonymous real applications Results are for both interpolation and extrapolation. Graphs show T over N.

experiments, is that the closer the observations are to the prediction target, the higher the accuracy. This, however, was our original assumption and the base for the curve fitting done by ADEPT.

As a rough comparison to white-box approaches results in [A.Snavely 2001] obtained with both application and machine modeling, show 97% and 81% accuracies for the CG benchmark on 32 and 64 nodes. For the same benchmark and numbers of nodes, our proposed method achieves more than 90%, and 82%. The accuracies for the very complex white-box approach presented in [G.Marin 2004] were about 90% for SP, about 90% for BT, and 80% to 90% for LU. For certain cases, the results show lower accuracy (e.g. 75% for LU). Our proposed method achieves accuracies of above 90% for SP, above 90% for BT, and above 80% for LU for the experiments shown (note that in our approach distance from observations matters), with the few exceptions mentioned above. Thus, our much cheaper and easier-to-apply approach provides almost the same accuracy and in some cases even better accuracy as the above white-box approaches.

3.9. Anomaly Detection

3.9.1. *General Approach for Detecting and Handling Anomalies*

Though real applications deviate from the Downey model to at least some extent, larger deviations are considered as "anomalous" behavior and must be detected by ADEPT. ADEPT detects candidates of anomalous behavior with an approach described below and then applies one of the two options for resolving them:

- Identification of anomalous individual points
- Recognition of typical patterns of irregular behavior

Anomaly candidate identification uses a fluctuation metric, defined as $R_i = \left(\left(t_i * n_i / n_{i+1}\right) / t_{i+1}\right) * \left(1 + \left(n_{i+1} - n_i\right) / n_{i+1}\right)$ for observation points *i* and *i+1*, which is applicable whether or not *T(1)* is provided. The expression $\left(\left(t_i * n_i / n_{i+1}\right) / t_{i+1}\right)$ expresses the ratio of projected runtime, assuming ideal relative speedup, vs. the measured runtime. This is how users may check scalability trends if *T(1)* is not available. However, ratios may fluctuate even for normal speedup curves if the distance between node counts in the available measurements varies significantly. Adjusting the metric to reflect relative distance between observations using $\left(1+\left(n_{i+1}-n_i\right)/n_{i+1}\right)$ removes such fluctuations.

We introduce a sensitivity factor, ε, which specifies the percentage of increase in *R* that will be ignored, considering that small fluctuations are normal. For any three observation points *i*, *i+1*, and *i+2*, if $R_{i+1} > (1+\varepsilon)R_i$, we flag Observations *i+1* and *i+2* as anomaly candidates.

Should points from the declining phase of the application be among the input, they can be detected unless being a single final point. The latter case cannot be handled by any black-box approach, since the point may be a declining-phase point or an anomaly and this uncertainty can be reported by ADEPT. In Figure 9 (bottom row, left) the interpretation of a declining phase is chosen, and no predictions are made for this point.

3.9.2. *Individual Anomalous Points*

After flagging the anomaly candidates, anomaly detection attempts to identify individual anomalous points causing fluctuation in the *R* curve. The following actions are taken:

- For each anomaly candidate, examine the overall *R* curve resulting from the removal of that point. Removing anomalous observations greatly decreases the fluctuation of the *R* curve, compared to removal of normal observations, thus identifying anomalous points. See Figure 9 for an example of an anomaly at 64 nodes, the corresponding *R* curve, and the two *R* curves resulting from removing each of the anomaly candidates. A minimum number of four input points is required to attempt detection of individual anomalous points.

- For anomalous point *l*, chosen from anomaly candidates *i* and *i+1*, calculate the magnitude of the deviation as $D_l = \min(10, (R_{i+1} - R_i)/\varepsilon)$.

Individual anomalous points and their corresponding magnitude of deviation are reported to the curve fitting component described in 3.4, which adjusts their weights as: $W_i^{'} = \max(0, W_i * (\theta - D_i)/\theta)$ to reduce the impact of the anomalous point on curve fitting. The deviation tolerance threshold *Θ* can be set to any value, where higher values meaning less sensitivity. The values for the two parameters $\varepsilon$and θ were set to 0.1 and 5, respectively, in our experiments. We found these values to be optimal in the detection of anomalies and setting of weights for anomalous points, for our experimental dataset. Based on the above description, the following is the algorithm used by the anomaly detection component:

1. For each pair of consecutive observations, calculate the R metric. The set of R metric values is called R.
2. Identify anomaly candidates: for each pair of consecutive values in R, mark observations i and i+1, if $R_{i+1} > (1 + \varepsilon)R_i$.
3. Remove each anomaly candidate, and recalculate the R metric values. If there are no more candidates, set weights according to above formula and stop.
4. If anomaly candidates still exist, apply one of the specific scalability patterns (as detailed in the next subsection).

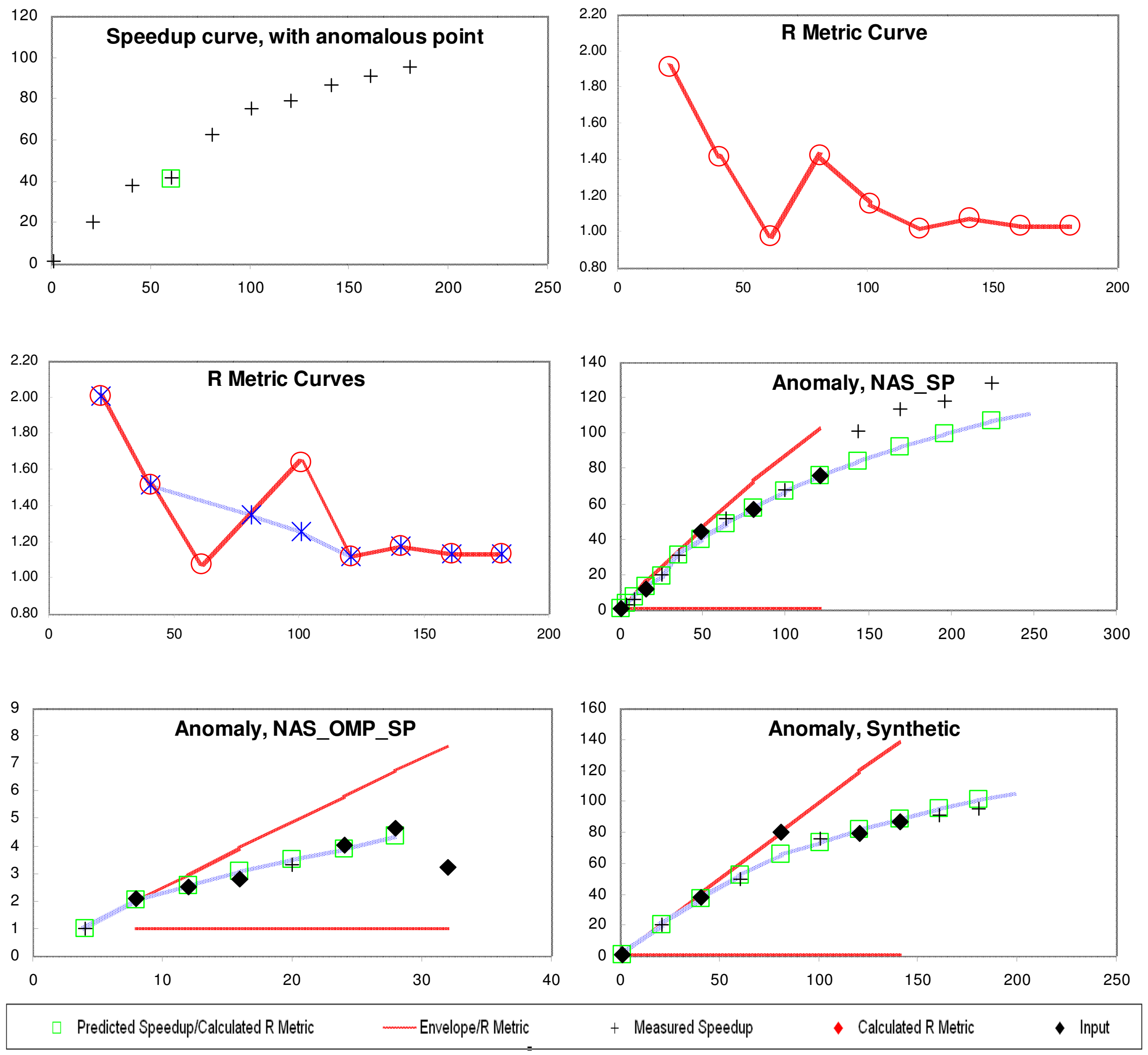


Figure 9. Detection and handling of individual anomalous points.
Top row shows a synthetic speedup curve with an anomalous point (boxed), and the associated *R* metric curve. Middle row shows the *R* metric curves resulting from removal of anomaly candidates. Middle row right and the bottom row show experiments integrating the anomaly detection step into ADEPT: Middle row right shows the speedup curve predicted by reducing the weight of anomalous point at 49 nodes. The bottom row shows the predicted speedup curve for NAS_OMP_SP by adjusting the weight of anomalous point at 16 nodes (left); and speedup curve predicted in spite of the anomalous input point at 80 nodes (right). Top row left, middle row right, and bottom row plot *S* over *N*, or *S* over number of threads for OMP. Top row right plots *R* metric over *N*.

During the testing of ADEPT for the NAS benchmarks, in several cases, anomalous points were identified and given reduced weight. See Figure 9 for two of the more easily distinguishable cases. For SP, the input point at 49 nodes was identified as having too high speedup, resulting in a prediction 10% lower than the actual value, and testing with a synthetic graph demonstrates identifying anomalous points without using *T(1)*, and how points which are too far off (measured speedup at 80 nodes being 20% too high) can be dropped, permitting good fits for other points.

3.9.3. *Specific Scalability Patterns*

Specific scalability patterns are detectable by the *R* metric curve. Different patterns can easily be defined. We currently detect and handle the following two important patterns:

- Stepwise scalability: Smaller data partitions per node often lead to enhanced performance if data fits into the next level of the memory hierarchy, potentially producing super linear speedup. The resulting stepwise scalability can be identified as a sharp spike in the *R* metric curve which is not improved with the removal of anomaly candidates. See Figure 10 for an example. We address the problem by multi-phase modeling, with one model instance per phase. For a single prediction target, the curve fitting step sets weight to zero for points not belonging to the same phase as the closest observation. A minimum of five input points over two phases are required to capture this pattern; fewer input points will not demonstrate such behavior.

- Specially optimized: Applications optimized, e.g. regarding communication, to run efficiently on certain numbers of nodes are recognized by having anomalous points with too high speedup at regular intervals. In this case, the regular anomalous points are considered as the most valid input and have their *D* value set back to zero, while all other points are discarded. Additionally, constraints are reported in regards to which are the feasible numbers of nodes to run the application on. Note that such application behavior is typically known by the user and could be specified as suggested for some adaptive job schedulers [W.Cirne

2003]. ADEPT permits automatic detection of behavior and constraints. A minimum number of nine input points is required for detection of this scalability pattern.

To test ADEPT's ability to handle specific scalability patterns, we constructed examples with synthetic data for each of the patterns mentioned above, see Figure 10. For the first stepwise case (OMP_FT), ADEPT identifies the change of program phase between 5 and 6 CPUs, and chooses the appropriate subsets of input points for the targeted prediction, resulting in excellent prediction accuracy. Similarly, ADEPT is capable of handling three-phase stepwise behavior. As shown in Figure 10 (Bottom row left), the application changes phase at 81 nodes and once again at 196 nodes. ADEPT identifies both phase changes and selects the correct subset of input points for each prediction target, providing highly accurate predictions.

The test case for specially optimized applications demonstrates that ADEPT fits and predicts only for nodes which are powers of two. Extension to other typical node allocations is straightforward.

3.10. Automated Reliability Judgment

Reliability judgment takes into account the placement of observation points, the maximum fitting error, and the existence of significantly different model instances which explain the input nearly equally well. The list of reliability problems, their indicators, and corresponding actions are presented in Table 3.

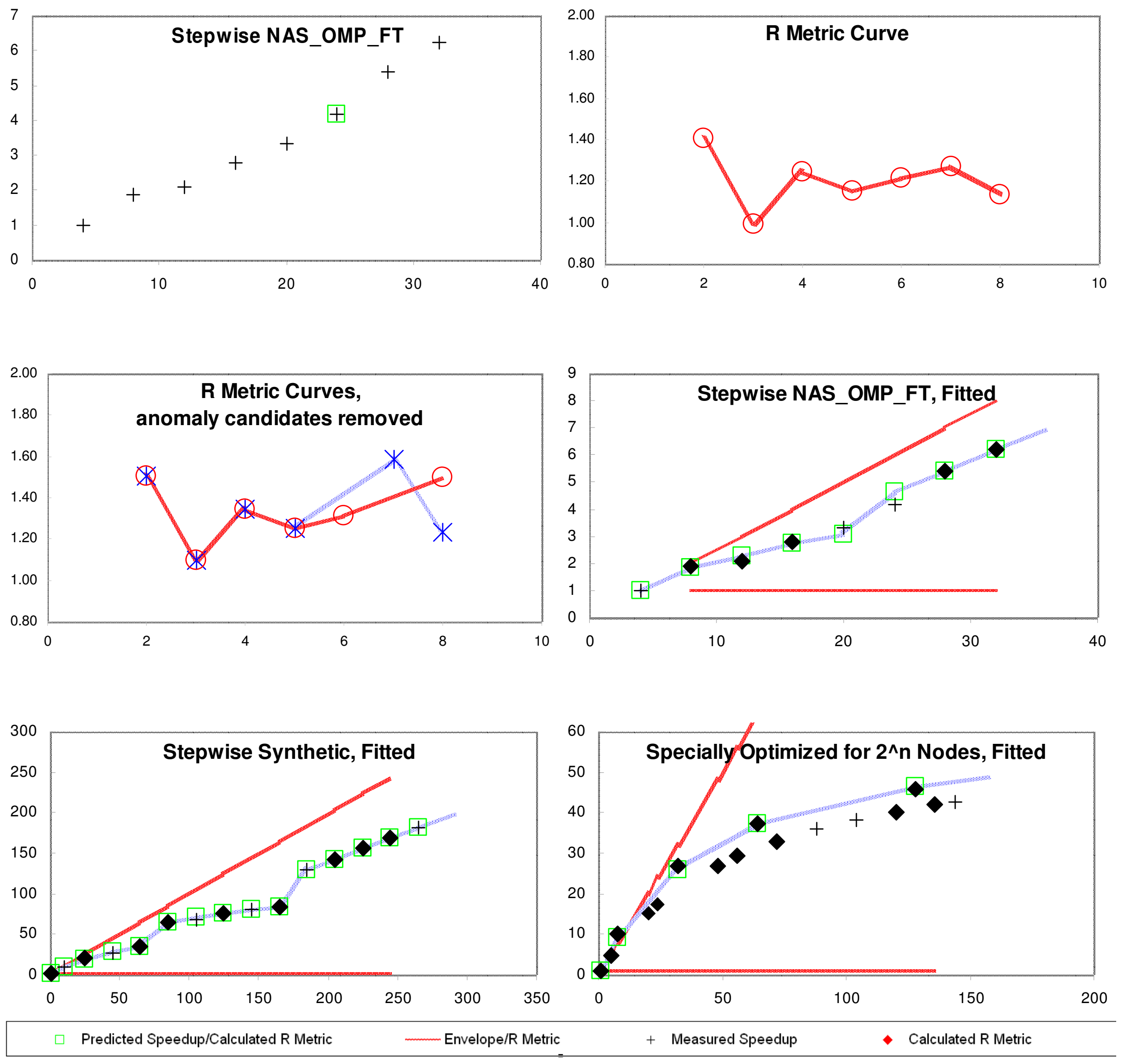


Figure 10. Detection and handling of specific scalability patterns.
Top row: Speedup curve for runtimes used as input (left), with step identified by a square symbol. R metric curve for all input points (right). Middle row: R metric curve with candidate points at 6 CPUs removed (stars) and at 7 CPUs removed (circles), (left); Resulting prediction from example in top row (right), Bottom row: additional example of stepwise speedup with three phases (left) and specially optimized application (right). Top row left and middle row right show S over number of threads, top row right and middle row left plot R over N, bottom row show S over N.

High fitting error is the simplest case. Input points are not identified as anomalous but experience large fitting errors (>10%). See LU in Figure 11 (top row left); the point at 32 nodes experiences 16% error in speedup (18% error in runtime). ADEPT

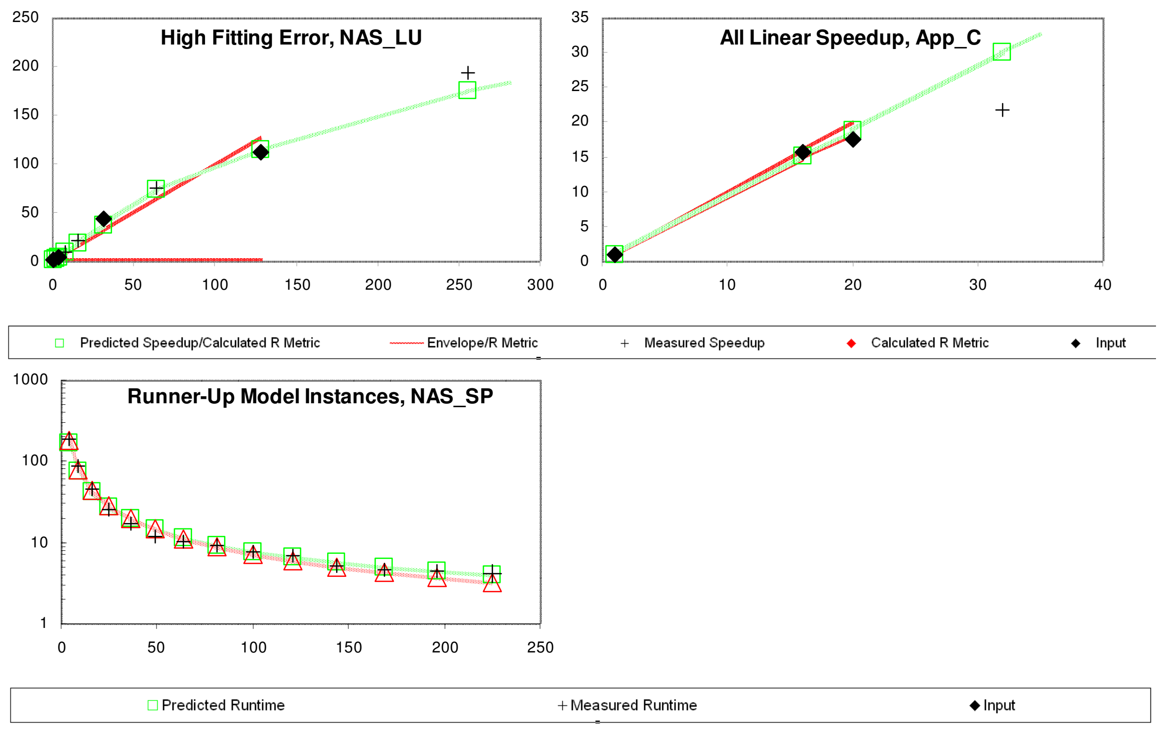


Figure 11. Reliability judgment

Results show high fitting error, all linear inputs, and runner up instances. Top graphs show S over N. Bottom graph shows T over N in log scale.

also detects if the model instances generated to perform prediction are of the low-variance class, and checks that at least one input point lies on the nonlinear section of the model. See App_C in Figure 11 (top row right) for an example where the input points all lie in the linear section. In this case, ADEPT suggests running on 105 nodes to collect further meaningful data.

The runner-up problem occurs when the data provided as input can be explained by at least two model instances with greatly different A. An example is shown in Figure 11 (bottom row), where the fitted model instance for prediction target at 49 nodes has a value of 700 for *A* parameter, and there is a runner-up instance with a value of 320 for *A*. This occurs because, as shown in the graph, the runtime values for the two model instances converge near the input points. The difference between runtime values of the two instances is less than 5% at input points of 16, 25, 36, and 81 nodes. Providing an additional input point at 225 nodes, where the two instances suggest runtime values

Table 3. The list of reliability problems, their indicators, and corresponding actions

Problem Description	Indicator	Action
Observation points all in linear section	Low variance model: $\forall (n_i, t_i) \in I : n_i < A$ High variance model has no linear section	Request additional observation on ≥*A* nodes
High fitting error: application deviates greatly from Downey model	$\max \left(\frac{\lvert actual - predicted \rvert}{actual} \right) > \delta$	Report problem
Multiple model instances with significantly different *A* values fit well (runner-up problem)	For model instances *i* and *j*: $WSSRE_i < WSSRE_j * 1.1$ where $A_i < A_j / 1.5$ or $A_i > A_j * 1.5$	Request additional observation (outside current range)

that differ by 20%, resolves the problem by identifying the runner-up instance as the best fit.

3.11. Performance Prediction: the Hypothesis

We have presented a black-box approach for predicting speedup and runtime of parallel applications. Our ADEPT predictor is both accurate and efficient by introducing an envelope derivation technique which constrains the search during model fitting and outperforms other model-fitting approaches. In our experiments with data from selected MPI and OpenMP NAS benchmarks and seven real applications, ADEPT showed high accuracy for both interpolative and extrapolative speedup/runtime prediction, even if not knowing the serial runtime. ADEPT delivers similar performance to that reported in the literature for white-box models if predicting for the same machine (see Section 3.8.1 for details of the comparisons) and is cheaper and suitable for large-scale use. ADEPT only requires a few observations and addresses practical problems of real applications. These are effectively handled by ADEPT via anomaly detection, using a fluctuation metric and automatic correction. Additionally, reliability judgment issues warnings if the prediction is uncertain and makes suggestions for further observations.

As outlined in the related works section, a key problem in the HPC area is the prediction of the relation between problem sizes and runtime/speedup of parallel applications. Solving this problem would allow more accurate resource allocation requests (by parallel applications) and decisions (by schedulers), which will contribute to obtaining the significant benefits of adaptive resource allocation. However, to the best of our knowledge, none of the black-box or gray-box prediction tools have the capability to predict runtime/speedup for both single problem size and across different problem sizes. ADEPT provides the foundation for addressing this more challenging problem in HPC, and we will next outline our proposed plan for the extension of ADEPT in this direction as it is in high demand and importance in both the literature and production environments, thus making ADEPT a more applicable tool. We form the working hypothesis for addressing the above challenge as follows: it is possible to construct a gray-box runtime/speedup predictor with the following requirements:

3.11.1. *Accuracy*

The predictor's accuracy is comparable to that of white-box methods, whenever such comparison is possible. The accuracy requirement applies to both cases of same problem size and different problem sizes. If using ADEPT as the foundation for the new predictor, the current capability in achieving white-box accuracy for the same problem size should not be negatively affected by the extension.

3.11.2. *Efficiency*

The predictor is applicable in a production environment, in the sense that 1) the requirements for using the predictor are close enough to black-box methods that the applicability is the same, and 2) user oradministrator intervention is not required except for providing data that can be reasonably expected to be already known to them.

3.11.3. *Robustness*

As with any production environment, there are always applications with anomalous behavior. The predictor needs to detect, and correct when possible, such anomalies, whether they are single points or form a special pattern of behavior in terms of runtime/speedup. In addition, the predictor needs to identify situations where obtained predictions are unreliable, and provide mechanisms for resolving such cases.

3.12. ADEPT Cross Problem Size Runtime and Speedup Prediction

We next outline the details regarding the extension of ADEPT to handle different problem sizes according to the requirements set forth by the hypothesis. Problem size is a domain-specific combination of input values and data structures passed to the parallel application, e.g. nodes of a graph, through any means, at compile time, runtime, or a combination of both. Our proposed extension to ADEPT treats all these details as a black-box method, i.e. does not require any information on them. The only additional required input is the association of a problem size identifier with each observation of runtime over a specific number of cores. Without loss of generality, we assume the following regarding prediction across problem sizes:

- Prediction is from a smaller to a larger problem size; this matches the typical use case, i.e. moving on to a larger instance of a problem once the smaller instance has been solved. We call the smaller problem size the base problem size, and the larger problem size the target problem size.

- A minimum of 4 or more observation points exist on the base problem size. This is a valid assumption and does not affect the applicability of ADEPT, as the move to the target problem size is expected only after having a reasonable number of observations on the base problem size.

- A minimum of 2 observation point exists on the target problem size.

Based on the above assumptions, we first describe the general case, i.e. no anomaly in the data. Details regarding anomaly detection and reliability judgment are specified in the following subsections. The key idea in translating behavior across problem sizes, thus making prediction across problem sizes possible is assuming similarity in behavior regardless of the problem size. The main shortcoming of such an assumption is ignoring the flat section of the scalability curve, but as described previously, this is an inherent limitation of the Downey model, and we address it through anomaly detection and reliability judgment. We consider similarity in behavior to demonstrate itself as relatively constant ratios among runtime values of two problem sizes across different numbers of nodes/cores, i.e. over different numbers of cores there is little fluctuation in the ratio of the runtimes of the two problem sizes. Although such notion of similarity exists in the literature [F.Nadeem 2006], ADEPT differentiates itself by going beyond simply taking such estimations as the actual runtime predictions. ADEPT uses these estimations as guiding points which are combined with actual observations of the target problem size runtime to provide both runtime and speedup predictions. It is important to note that existing work lack the latter capability, i.e. speedup prediction. More specifically, ADEPT uses the following algorithm in order to predict runtime and speedup across problem sizes:

1. Find the smallest number of nodes for which observations of runtime exist for both base and target problem sizes. Calculate the ratio of runtimes as follows:

$$R_{base,target} = T_{target,n0} / T_{base,n0}$$

Where n0 is the smallest number of nodes/cores for which runtime observations exist for both base and target problem sizes, Ttarget,n0 is the runtime of the target problem size at n0, and Tbase,n0 is the runtime of the base problem size at n0 nodes/cores.

2. Add estimated runtime values as guiding points to the target problem size

   a. Form the set of number of nodes/cores for which there are observations on the base problem size but not the target problem size:

   $$N_{est} = \{ n_i \mid n_i \in T_{base} \wedge n_i \notin T_{target} \}$$

   b. Form the set of estimated runtimes for the target problem size, where $t_{base,i}$ is the runtime of the base problem size at i cores.

   $$R_{est} = \{ (n_i, ti) \mid n_i \in N_{est} \wedge t_i = t_{base,i} * R_{base,target} \}$$

3. Use the following set as input to the predictor component:

   $$T'_{target} = T_{target} \cup R_{est}$$

4. Perform model fitting on the the T'_{target} set of nodes and runtime values, as described in details in Section 3.5.

We next provide experimental results using the proposed extension method.

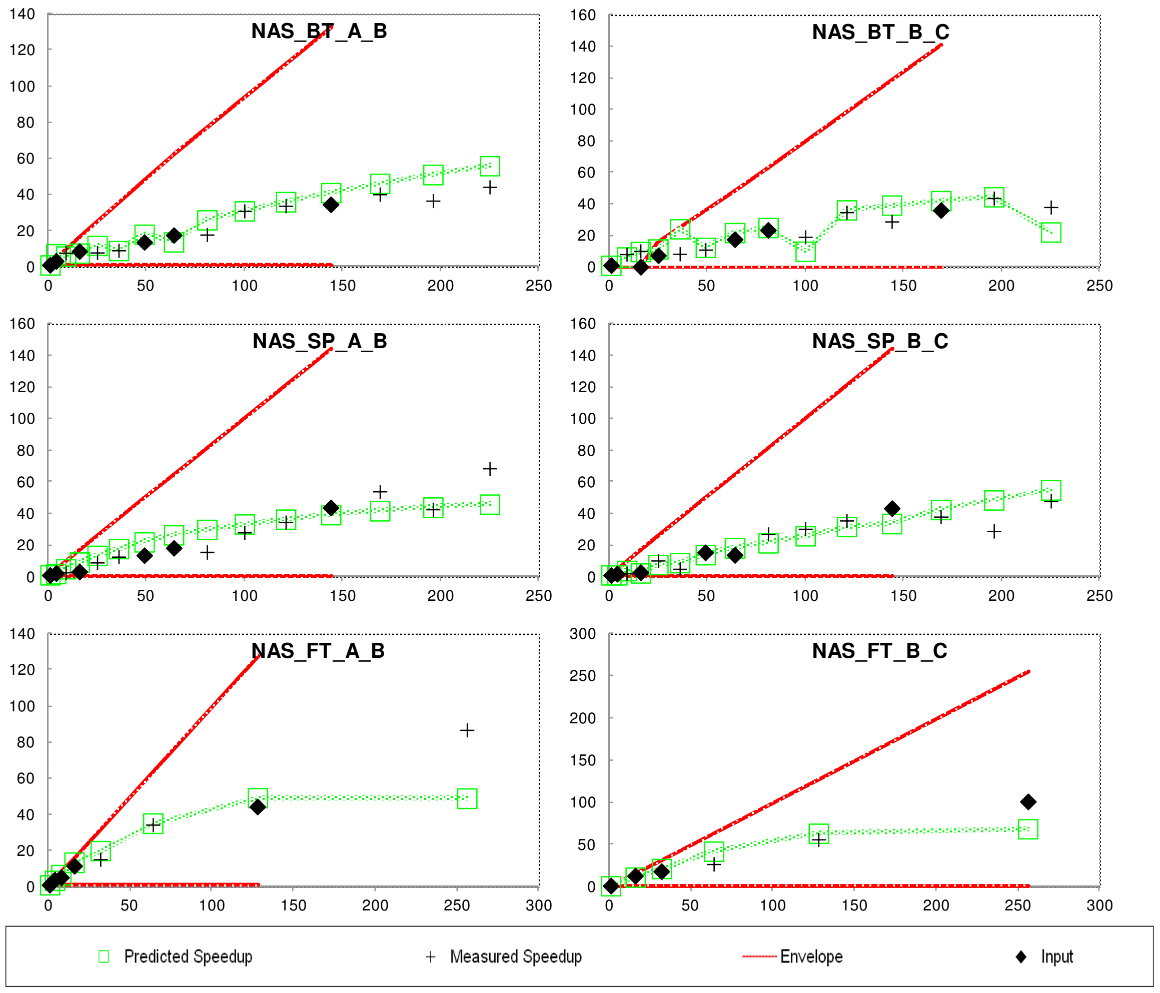


Figure 12. Speedup prediction results across problem sizes, MPI implementations of NAS BT, FT, and SP. Results are for both interpolation and extrapolation. Graphs show S over N.

3.13. *Experimental Results: Performance Prediction across Problem Sizes*

3.13.1. *Speedup Prediction across Problem Sizes*

This section details the experimental results for the proposed extension method. NAS benchmarks BT, CG, FT, LU, and SP have been used for the experiments. We have run classes A, B, and C of each benchmark. As in previous experiments, we use the following accuracy metric:

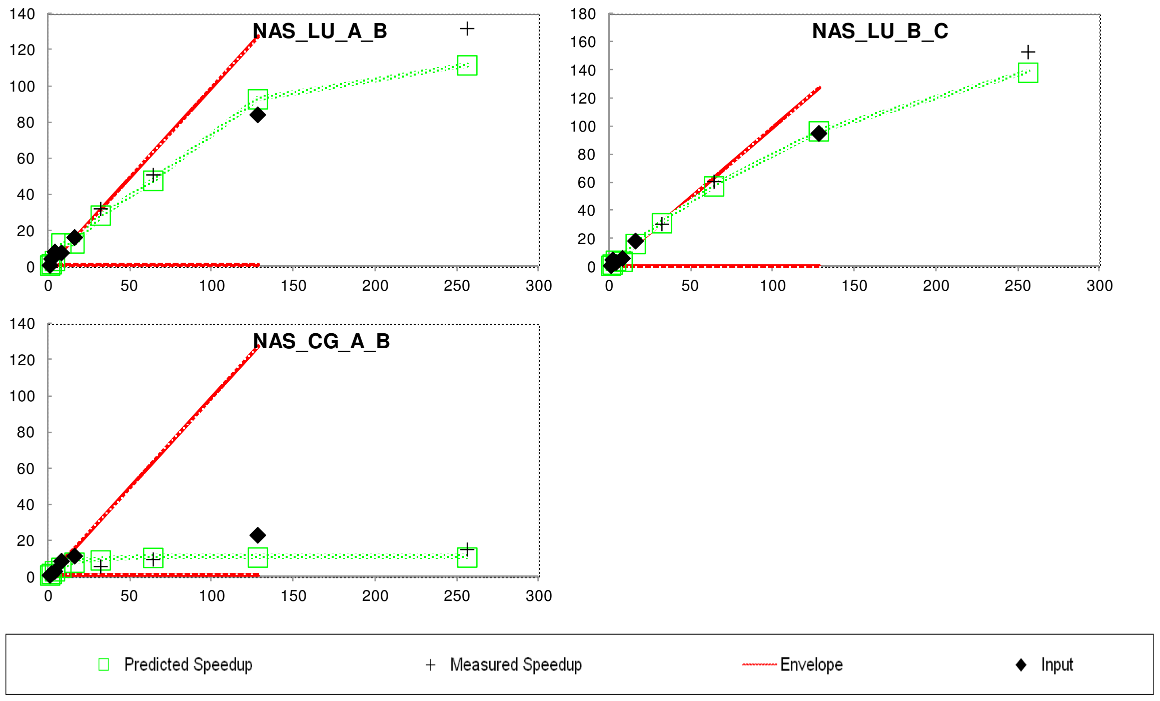


**Figure 13. Speedup prediction results across problem sizes for NAS CG and LU.
Results are for the MPI implementation of NAS CG, and LU, both interpolation and extrapolation.
Graphs show S over N.**

$$Accuracy = 100 - (100 * |Actual - Predicted|/Actual)$$

Figure 12 and Figure 13 show the experimental results for speedup prediction. Accuracy of the predictions is generally above 70%, with many predictions having 90% or more accuracy. These results are achieved using only the 2 smallest numbers of nodes on the target problem size as input, and 4 or 5 input points on the base problem size of the target benchmark. There are certain benchmarks, in particular CG, which show several points with inaccuracies. We will provide details in the following sections on how ADEPT handles such cases through anomaly detection and reliability judgment. To demonstrate the envelopes derived by ADEPT for each benchmark, we combine all the envelopes into one using the method described in Section 3. The drawn envelope is based on the predicted T(1), as ADEPT does not require T(1) to be available for a target application.

In summary, prediction accuracies are generally above 70% using the proposed method. There are a few exceptions; LU class C at 8 cores has low (~ 50%) prediction accuracy which we attribute to LU class B having several anomalous points at 4, 8, and 16 cores, thus not providing useful input data to ADEPT for this prediction target. LU class B, when used as the prediction target, also shows approximately the same low prediction accuracy for 8 cores, which we attribute to both anomalous and dissimilar behavior at 4, 8, and 16 nodes for both classes A and B of LU. Benchmark CG shows approximately 52% prediction accuracy at 32 cores with classes B and C as base and target problem sizes, respectively, due to the anomaly at 32 nodes in class C.

3.13.2. Experimental Results: Runtime Prediction across Problem Sizes

Next, we demonstrate the performance of ADEPT at runtime prediction. Results are shown in Figure 14 and Figure 15. It should be noted that the runtime predictions for several of the benchmarks are so accurate that measurements and predictions overlap on the runtime curve. The number of runtime measurements used as input was either four or five on the base problem size and exactly two for the target problem size for all experiments. Since the runtimes span a long range of values, the runtime axis has logarithmic scale to better separate predictions and the observation points. We show a single graph per benchmark, which corresponds to an experiment that did not use *T(1)* as input, since the provision of *T(1)* as input did not result in any major improvement in the runtime prediction accuracy. Accuracies obtained for runtime prediction, excluding *T(1)*, were generally above 70%, with the major exceptions being FT at 256 cores on both classes B and C, SP class C at 36 cores, SP class B at 225 cores, and LU class C at 128 cores. Regarding the accuracies of interpolations vs. extrapolations, considering the only actual observations for the target problem size are the two smallest numbers of cores available, all predictions can be considered extrapolation for cross problem size

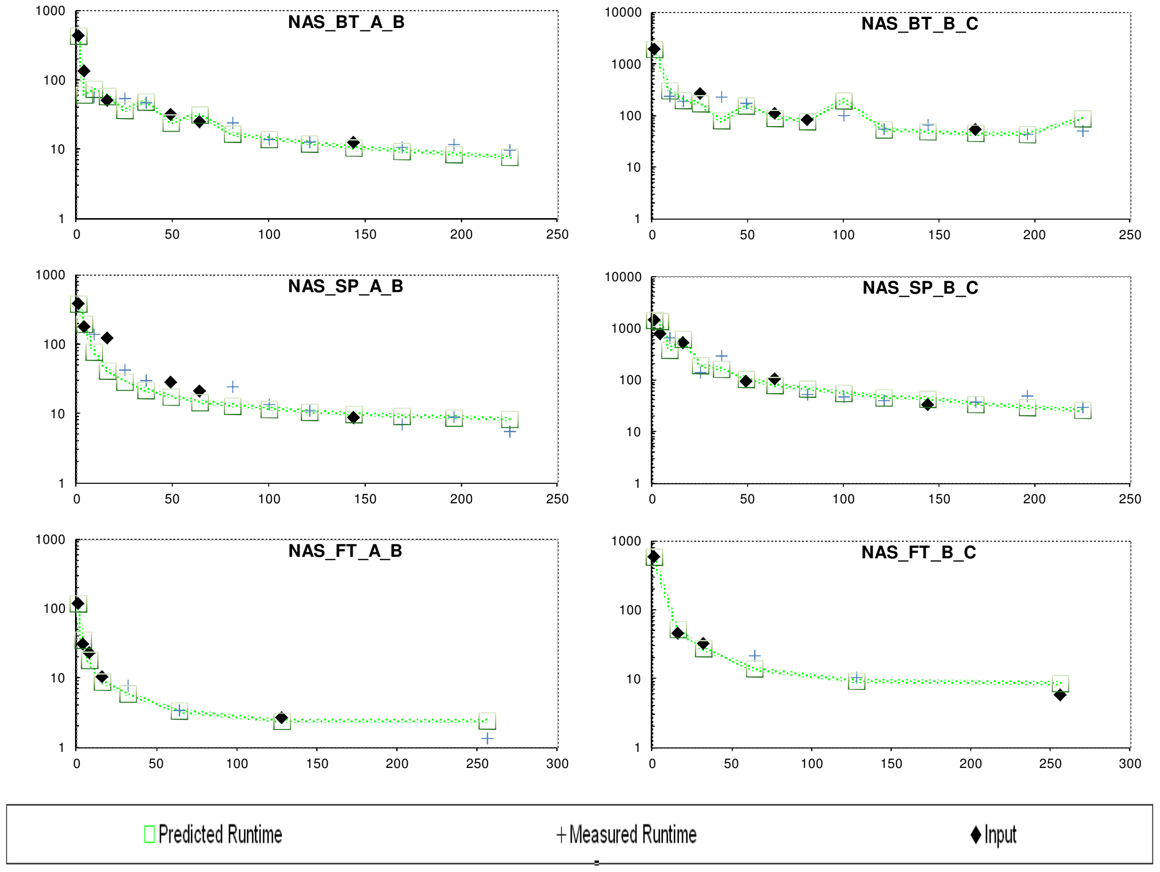


Figure 14. Runtime prediction across problem sizes, NAS BT, SP, and FT.
Results are shown for both interpolation and extrapolation. Graphs show T over N. The title of each graph speicifes in order the benchmark, the base problem size, and the target problem size.

predictions. Using the base problem size observation points to distinguish between interpolative and extrapolative predictions shows no significant difference between accuracies.

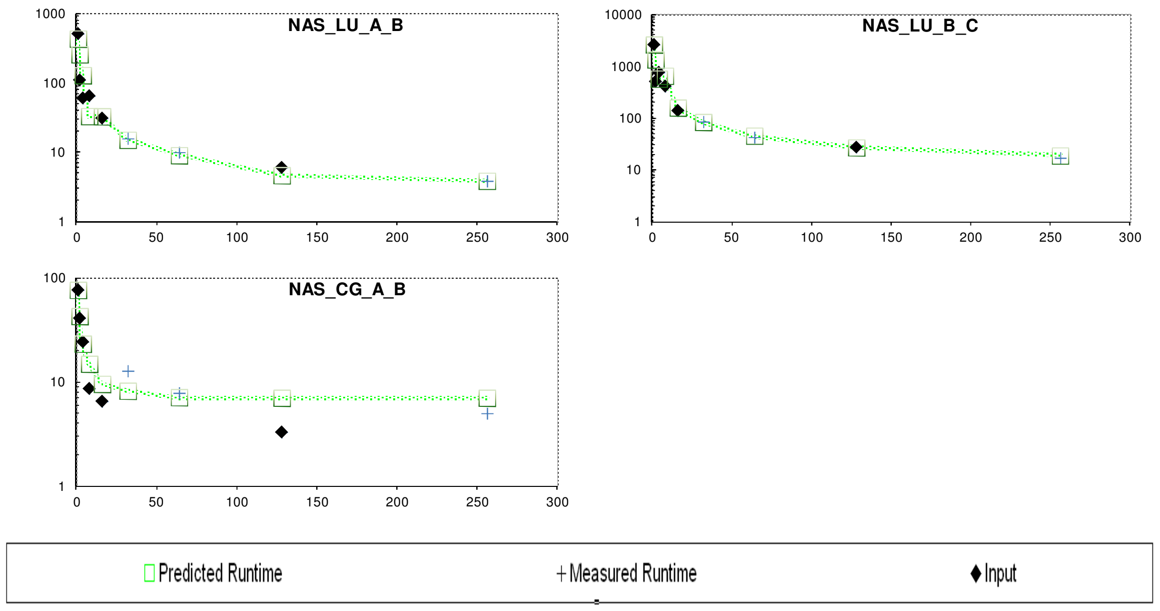


Figure 15. Runtime prediction results across problem sizes, NAS CG and LU
Results are shown for both interpolation and extrapolation. Graphs show T over N. The title of each graph speicifes in order the benchmark, the base problem size, and the target problem size.

3.14. General Approach for Detecting and Handling Anomalies over Different Problem Sizes

As described earlier, ADEPT handles anomalies in target applications' behavior, and corrects predictions accordingly when it is possible to do so considering the available information.

Prediction across problem sizes introduces a new challenge for anomaly detection: there are not enough observations on the target problem size. Therefore, there are 2 possible directions ADEPT can take to handle anomalies: 1) assume anomalies at the same numbers of cores for the base and target problem sizes, and 2) assume individual anomalous points to be specific to a problem size. With the former assumption, ADEPT needs to assign a higher weight to points estimated based on the individual anomalous point of the base problem size, when making predictions for the same number of nodes in the target problem size. It will also mean a lower weight should be assigned to such estimated points when predicting for all other points of the target problem size. The latter assumption would require ADEPT to reduce the effect of individual anomalous points of the base problem size on every prediction point of the target problem size. ADEPT corrects this assumption if new observations on the target problem size indicate it to be false, i.e. if a new observation for the target problem size shows an individual anomaly at the same number of nodes as the base problem size. Figure 16 (left) shows an example in which both problem sizes have anomalous points at the same number of nodes. For the second assumption, Figure 16 (right) shows problem sizes with anomalous points at different numbers of nodes.

As with prediction for a single problem size, large deviations from the Downey model are considered as “anomalous” behavior and need to be detected by ADEPT. ADEPT detects candidates of anomalous behavior across problem sizes using the R metric described previously and then applies one of the two options for resolving them:

- Identification of anomalous individual points
- Recognition of typical patterns of irregular behavior

We describe the former option next. Handling of typical patterns of irregular behavior is the same for single problem size and cross problem size predictions.

3.14.1. Individual Anomalous Points

ADEPT first performs anomaly detection on the observation points of the base problem size. It then attempts to identify individual anomalous points causing fluctuation in the *R* curve. The following actions are taken:

- As with the case of a single problem size, ADEPT examines the overall *R* curve resulting from the removal of each anomaly candidate observation. Removing

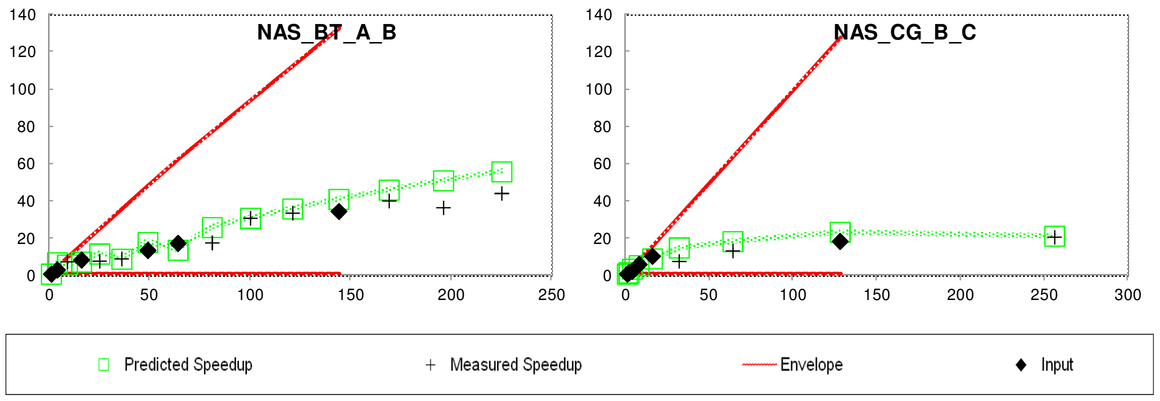


Figure 16. Detection and handling of individual anomalous points across problem sizes. **Shows experiments integrating the cross problem size anomaly detection step into ADEPT: speedup curve predicted by reducing the weight of anomalous point at 25 nodes of BT class A on prediction for BT class B(left); predicted speedup curve for NAS CG benchmark class C by adjusting the weight of anomalous point at 32 nodes of CG class B (right). Charts plot S over N.**

anomalous observations greatly decreases the fluctuation of the *R* curve, compared to removal of normal observations, thus identifying anomalous points.

- For each anomalous point *l*, chosen from anomaly candidates *i* and *i+1* from the set of observation points of the base problem size, calculate the magnitude of the deviation as $D_l = (R_{j+1} - R_j)/\varepsilon$.

Individual anomalous points and their corresponding magnitude of deviation are reported to the cross problem size prediction module. ADEPT then determines whether to use each of the estimated observation point based on the magnitude of deviation of the corresponding base problem size observation. There are two cases:

- The anomalous point in the base problem size does not correspond to the current prediction target. ADEPT will use the estimated observation based on this point only if the magnitude of deviation is below a certain threshold called γ.

- The anomalous point corresponds to the current prediction target. ADEPT will use the estimated observation based on this point as input for the current prediction target.

During the testing of ADEPT for predicting cross problem sizes for the NAS benchmarks, in several cases, anomalous points were identified and handled using the above method. In the experiments, we set γ to 5. We found this value to be optimal in detection of anomalies. See Figure 16 for two of the more easily distinguishable cases. For CG class C when the base problem size is CG class B, the input point at 32 nodes was identified as having too low a speedup, i.e. runtime was too high. Cross problem size anomaly detection and handling removes the estimated observation that is based on this point from the set of input points for class C, resulting in prediction accuracies of 73% and higher for CG class C. When predicting for BT B using BT A as the base class, observations at 25 and 49 nodes are marked as having too low speedups, i.e. too high runtime. ADEPT handles these anomalous observation points by removing the corresponding estimated observation points, achieving generally above 70% prediction accuracy for BT class B when using A as the base class.

3.15. Reliability Judgment across Problem Sizes

Reliability judgment across problem sizes addresses all the 3 cases of unreliable predictions handled for a single problem, i.e. high fitting errors, existence of runner-up, and all linear-section observations. In addition, cross problem size reliability judgment detects and handles a scenario which is only applicable to multiple problem sizes: two problem sizes differing significantly enough in behavior to be considered 2 different applications.

3.16. Summary

In this chapter we described in detail our proposed performance prediction tool, ADEPT. We used ADEPT to prove our hypothesis that a gray-box tool can provide accurate performance predictions for both a single problem size and across multiple problem sizes of a parallel application, while staying applicable, robust, and efficient. Our overall goal was scalability prediction (in the sense of strong scaling), on both multi-core SMP servers and clusters, which is practically feasible for production environments. The gray-box nature of ADEPT, which uses only a single string input as indication of problem size in addition to the black-box observation of the target application, enables production use. ADEPT is based on the Downey model, shown to capture simplified behavior of parallel applications very well. ADEPT introduced the use of Downey model as a predictive model by addressing challenges set by real parallel applications, i.e. not fully conforming to the model and/or reliability of the predictions. With the development of ADEPT prediction tool, we pursued the following detailed goals:

- Achieve high prediction accuracy, while requiring only few observations (typically 3 to 4).
- Achieve accurate predictions for a) single problem size and b) multiple problem sizes of the target parallel application
- Provide a computationally efficient approach for deriving the model instance.
- Identify cases where the application does not fully conform to the Downey model as anomalies, with automatic correction and multi-phase modeling for individual irregular points and typical patterns.
- Perform reliability judgment which recognizes unsuitable observation layout and proposes placement ranges of additional observations.

Experiments with the NAS benchmarks and seven real applications demonstrated the efficiency and prediction quality of ADEPT in handling normal cases and anomalies. We obtained generally above 80% prediction accuracy for a single problem size and above 70% accuracy for cross problem size predictions using only the two smallest numbers of cores available on the target problem size, even in cases with anomalies and for predictions which extrapolate for more than twice the number of nodes that were used in the closest observation. The experiments also demonstrate the effectiveness of reliability judgment.

CHAPTER 4

Conclusion and Future Work

4.1. Conclusions

Performance prediction is the task of providing an estimation of the performance of an instance of an application. Having accurate predictions regarding the scalability and runtime of applications can potentially improve the performance of job schedulers significantly. However, such predictors have not become practical yet for a wide variety of reasons, among which are the requirements of existing tools including the need for user and/or administrator intervention and OS-level support.

In this dissertation, we presented an inexpensive, highly applicable performance prediction tool called ADEPT (acronym for Automatic Downey-based Envelope-constrained Prediction Tool). We set the following goals for ADEPT:

- Achieve high prediction accuracy, while requiring only few observations (typically 3 to 4).
- Provide a computationally efficient approach for making predictions.
- Identify cases where the application does not fully conform to the Downey model as anomalies, with automatic correction and multi-phase modeling for individual irregular points and typical patterns.
- Perform reliability judgment which recognizes unsuitable observation layout and proposes placement ranges of additional observations.
- Handle performance prediction across different problem sizes of an application

To address these challenges, ADEPT employs a novel approach that combines: envelope-derivation technique which constrains the search for the best-fitting model instance; a special metric for detection of anomalies; and special pattern handling for cases like super-linear speedup.

Having completed the requirements regarding prediction accuracy, anomaly detection and correction, and handling of issues regarding reliability of predictions, we next hypothesized that ADEPT can be extended to address the last challenge from the above list, i.e. prediction across problem sizes. We extended ADEPT to perform highly accurate predictions for different problem sizes of the same application. In this extension, ADEPT maintained its applicability, i.e. we did not introduce requirements with a different nature but only expected a few observations on the previous problem sizes of the application, as we expect a user to move to larger problem sizes once the behavior of the application on a smaller problem size is available.

Experiments using ADEPT on observations from both single problem size and multiple problem sizes of the NAS benchmarks and several practical applications used on SHARCNET clusters demonstrated highly accurate predictions made by ADEPT. Predictions for a single problem size, when compared to several complex and expensive white-box methods, are either higher or comparable in terms of accuracy. The following is the summary of the experiments:

- For single problem size, prediction accuracies were generally above 70% using the proposed method, with a few exceptions which are specified in Sections 3.8.1 and 3.8.2.
- For prediction across problem sizes, accuracies were generally above 70% and many predictions having accuracies above 90%, and only a few exceptions which are specified in Sections 3.13.1 and 3.13.2.

- Cross-validation experiments done for both single problem size and across problem size predictions did not indicate high sensitivity to any single normal or anomalous observation in general.
- ADEPT handles both individual anomalies and specific scalability patterns using the R-metric method presented in Section 3.9.
- ADEPT correctly identifies unreliable predictions and recommends adding more observations if such observations could potentially result in reliable predictions, see Section 3.10 and 3.15 for details.

4.2. Future Work

ADEPT addresses several key challenges of performance prediction and this can potentially make it a highly applicable tool for both job schedulers and users of HPC applications. Requiring very few observations of the target application’s behavior, eliminating the need for OS-level support and interference from user and administrator to obtain white-box details on the application, e.g. interconnect usage, are among the key characteristics of ADEPT.

However, in order for ADEPT to become an even more applicable tool, there are several directions which we intend to pursue. The main one is performance prediction across different hardware, including CPU, cache memory, and interconnects. As we mentioned in Section 2.8.1, despite the existence of several publications, there are still many challenges to be addressed in this area. Another potential direction is extending ADEPT to handle platforms different in terms of software, e.g. different operating systems and different implementations of MPI. Such an extension needs to be aware of and account for interactions between different software components, the hardware, and the target application. Predicting across different platforms is especially important as users may need to move their application to other platforms which differ in terms of hardware and/or software and need an estimation of the performance of their application before arranging such a move which can be very costly in terms of both time and resources. A performance prediction tool will be potentially very valuable if it can address such scenarios while maintaining the applicability and low cost that ADEPT offers. Another potential extension for ADEPT is the capability to handle heterogeneous environments, e.g. grids. As chapter 2 presents in detail, many performance prediction methods operate in grid environments. However, to the best of our knowledge, there are no performance prediction tools with the same characteristics as ADEPT in such environments. To be applicable to grids, ADEPT needs to 1) be able to characterize a grid environment in terms of its effects on a parallel application, 2) be able to translate performance of a parallel application within a grid, i.e. when using different resources on the grid. These requirements need support from the grid, in terms of providing details on the runtime environment of each run of a target application, as well as some model of all the runtime environments offered by the grid. Applications relying on both CPUs and GPUs to run form another heterogeneous environment as a potential extension for ADEPT. One challenge in such environments is that the number of cores is potentially a multi-dimensional variable, as different numbers of CPU cores and GPU cores could affect the application differently.

REFERENCES/BIBLIOGRAPHY

[S. Achour 2011] S. Achour, M. Ammar, B. Khmili, and W. Nasri, MPI-PERF-SIM: Towards an Automatic Performance Prediction Tool of MPI Programs on Hierarchical Clusters, 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing, 2011, pp. 207-211.

[L. Adhianto 2006] Laksono Adhianto and Barbara Chapman. 2006. Performance Modeling of Communication and Computation in Hybrid MPI and OpenMP Applications. In Proceedings of the 12th International Conference on Parallel and Distributed Systems - Volume 2 (ICPADS '06), Vol. 2. IEEE Computer Society, Washington, DC, USA, pp. 3-8.

[S.R.Alam 2006] S. Alam, and J. Vetter, A Framework to Develop Symbolic Performance Models of Parallel Applications, 5th International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS 2006), held in conjunction with IPDPS 2006, 8 pp.-

[M.F.Arlitt 2000] M.F. Arlitt, "Characterizing Web User Sessions," ACM SIGMETRICS Performance Evaluation Rev., vol. 28, no. 2, pp.50-63, 2000.

[D.H.Bailey 1995] D.H. Bailey, T. Harris, W.C. Saphir, R.F. Van der Wijngaart, A.C. Wood, and M. Yarrow. The NAS Parallel Benchmarks 2.0, NAS Technical Report NAS-95-020,NASA Ames Research Center, Moffett Field, CA, 1995.

[K.J.Barker 2009] Kevin J. Barker, Kei Davis, Adolfy Hoisie, Darren J. Kerbyson, Michael Lang, Scott Pakin. Using Performance Modeling to Design Large-Scale Systems. In IEEE Computer, 42 (11): November 2009, pp. 42-49.

[B.Barnes 2008] B. Barnes, B. Rountree, D. K. Lowenthal, J. Reeves, B. R. de Supinski, and M. Schulz, "A Regression-Based Approach to Scalability Prediction," in International Conference on Supercomputing, Jun. 2008, pp. 368-377.

[B.Barnes 2010] Using Focused Regression for Accurate Time-Constrained Scaling of Scientific Applications, Brad Barnes, Jeonifer Garren, David K. Lowenthal, Jaxk Reeves, Bronis R. de Supinski§, Martin Schulz, and Barry Rountree.

[L.Barsanti 2006] L. Barsanti and A.C. Sodan, Adaptive Job Scheduling Strategies via Predictive Job Resource Allocation, Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), in conjunction with ACM SIGMETRICS, Saint-Malo / France, Springer, June 2006.

[L.Cai 2003] L. Cai, R. L. Malmberg, and Y. Wu. Stochastic modeling of rna pseudoknotted structures: A grammatical approach. In Proceedings of ISMB'03 and Bioinformatics 19(s1), pages i66–i73, 2003.

[L.Carrington 2005] L. Carrington, M. Laurenzano, A.Snavely, R. Campbell, L. Davis, How Well Can Simple Metrics Represent the Performance of HPC Applications?, SC 05 , Seattle, WA, November 2005

[L.Carrington 2003] L. Carrington, A. Snavely, X. Gao, and N. Wolter, A Performance Prediction Framework for Scientific Applications, *Proc. ICCS Workshop on Perf. Modeling & Analysis (PMA)*, Melbourne, Australia, June 2003.

[M. Casas 2008] Marc Casas, Rosa M. Badia, Jesus Labarta, Prediction of Behavior of MPI Applications, IEEE Cluster 2008, September 2008.

[S.H.Chiang 1996] S.-H. Chiang and M.K. Vernon. Dynamic vs. Static Quantum-Based Parallel Processor Allocation. *Proc. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, Springer, LNCS 1162, May 1996.

[Cirne 2001] Cirne, Walfredo and Fran Berman. "A comprehensive model of the supercomputer workload." Proceedings of IEEE 4th Annual Workshop on Job Scheduling Strategies for Parallel Processing. Cambridge, MA. 2001.

[W.Cirne 2003] W. Cirne and F. Berman. When the Herd is Smart: Aggregate Behavior in the Selection of Job Request. IEEE Trans. on Parallel and Distributed Systems, 14(2), Feb. 2003, pp. 181-192.

[J.Corbalan 2005] J.Corbalan, X.Martorell, J.Labrta. Performance-Driven Processor Allocation .IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. ISSN: 10459219 Vol: 16 pp: 599-611 July 2005.

[B.F.Cornea 2011] B.F. Cornea, J. Bourgeois, T.T. Nguyen, D. El-Baz, Performance Prediction in a Decentralized Environment for Peer-to-Peer Computing, in IEEE International Parallel & Distributed Processing Symposium (IPDPS), 2011, pp. 1618-1626.

[M.Curtis-Maury 2005] M. Curtis-Maury, T. Wang, C. Antonopoulos, and D. Nikolopoulos. Integrating Multiple Forms of Multithreaded Execution on mult-SMT System: A Study with Scientific Applications. Proc. Internat. Conf. on Quantitative Evaluation of Systems (QUEST), 2005.

[K.Davis 2009] Kei Davis, Kevin Barker, Darren J. Kerbyson. Performance Prediction via Modeling: A Case Study of the ORNL Cray XT4 Upgrade. In Parallel Processing Letters, 19 (4): December 2009.

[J.Delgado 2010] Javier Delgado, S. Masoud Sadjadi, Hector Duran, Marlon Bright, and Malek Adjouadi. Performance prediction of weather forecasting software on multicore systems. In Proceedings of the 24th IEEE International Parallel & Distributed Processing Symposium (IPDPS-2010), 11th Parallel and Distributed Scientific and Engineering Computing (PDSEC) workshop, Atlanta, Georgia, April 2010.

[A.Deshmeh 2009] A. Deshmeh, J. Machina, and A.C. Sodan, ADEPT Black-box Speedup and Runtime Predictor, Technical Report 09-018, University of Windsor, 2009.

[A.Deshmeh 2010] Deshmeh, A. Machina, J. Sodan, A., ADEPT scalability predictor in support of adaptive resource allocation, In 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS), April 2010, pp. 1-12.

[Dimemas 1997] J. Labarta, S. Girona, and T. Cortes, Analyzing scheduling policies using Dimemas, Parallel Comput. 23, 1-2 (April 1997), pp. 23-34.

[A.Downey 1997] Allen Downey, Predicting Queue Times on Space Sharing Parallel Computers, In Proceedings of the 11th International Symposium on Parallel Processing (IPPS ’97), 1997, pp. 209-218.

[A.Downey 1997 Model] A.B. Downey. A Model for Speedup of Parallel Programs. Technical Report CSD-97-933. UC Berkeley, 1997.

[R.Duan 2009] Rubing Duan, Farrukh Nadeem, Jie Wang, Yun Zhang, Radu Prodan, and Thomas Fahringer. 2009. A Hybrid Intelligent Method for Performance Modeling and Prediction of Workflow Activities in Grids. In Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09). IEEE Computer Society, Washington, DC, USA, 339-347.

[A.Duran 2008] A. Duran, J. Corbaln, and E. Ayguad, An adaptive cut-off for task parallelism. In Proceedings of the 2008 ACM/IEEE conference on Supercomputing (SC '08). IEEE Press, Piscataway, NJ, USA, 2008, Article 36 , 11 pages.

[T.Fahringer 2000] Thomas Fahringer, A. Pozgaj, Hans Moritsch, J. Luitz: Evaluation of P3T+: A Performance Estimator for Distributed and Parallel Applications. IPDPS 2000: 229-234

[U.Farooq 2009] Farooq, U., Majumdar, S., Parsons, E., “Achieving Efficiency, Quality of Service and Robustness in Multi-Organizational Grids”, Journal of Systems and Software (Special Issue on Software Performance), Vol. 82, Issue: 1, January 2009, pp. 23-38.

[D.G.Feitelson 1996] D.G. Feitelson, “Packing Schemes for Gang Scheduling,” Proc. IPPS Workshop Job Scheduling Strategies for Parallel Processing (JSSPP '96), D.G. Feitelson and L. Rudolph, eds., pp. 89-110, 1996.

[D.G.Feitelson 2007] D.G. Feitelson, “Locality of Sampling and Diversity in Parallel System Workloads,” Proc. 21st ACM Int'l Conf. Supercomputing (ICS '07), pp. 53-63, June 2007.

[D.G.Feitelson 2008] D. G. Feitelson, “Looking at data”. In 22nd Intl. Parallel and Distributed Processing Symp., Apr 2008.

[D.Ferrari 1984] D. Ferrari, “On the foundation of artificial workload design”. In SIGMETRICS Conf. Measurement & Modeling of Comput. Syst., pp. 8–14, Aug 1984.

[F.Freitag 2001] F.Freitag, J. Corbalán, J. Labarta. A Dynamic Periodicity Detector: Application to Speedup Computation. 15th International Parallel and Distributed Processing Symposium (IPDPS'2001). Proceedings of the 15th International Parallel and Distributed Processing Symposium. ISBN 0-7695-0990-8. San Francisco, USA. April 2001.

[A.V.Germund 2003] A. van Gemund. Symbolic performance modeling of parallel systems. IEEE Transactions on Parallel and Distributed Systems, 14(2), 2003.

[D.Ghosal 1991] D. Ghosal, G. Serazzi, and S.K. Tripathi, The Processor Working Set and its Use in Scheduling Multiprocessor Systems, *IEEE Trans. on Software Engineering*, 17(5), May 1991, pp. 443-453.

[R.Gibbons 1997] Richard Gibbons, A Historical Application Profiler for Use by Parallel Schedulers, Lecture Notes on Computer Science, 1997, pp. 58-75.

[C.Glasnerlow 2011] Christian Glasnerlow, and Jens Volkerta, Adaps – A three-phase adaptive prediction system for the run-time of jobs based on user behaviour, Journal of Computer and System Sciences, Volume 77, Issue 2, March 2011, Pages 244-261, Adaptivity in Heterogeneous Environments.

[F.Guim 2005] F. Guim, A. Goyeneche, J. Corbalan, J. Labarta, G. Terstyansky. Grid computing performance prediction based in historical information. First CoreGRID Integrated Research in Grid Computing Workshop. November 2005.

[F.Guim 2008] I. Rodero, F. Guim, J. Corbalan et al., "The Grid Backfilling: a Multi-Site Scheduling Architecture with Data Mining Prediction Techniques," Grid Middleware and Services, pp.137-152, 2008.

[S.D.Hammond 2009] S.D. Hammond, J.A. Smith, G.R. Mudalige, and S.A. Jarvis, Predictive Simulation of HPC Applications, Proc. IEEE 23rd International Conference on Advanced Information Networking and Applications (AINA-09), 2009, pp. 33-40.

[J.He 2011] J. He, A.E. Snavely, R.F. Van der Wijngaart, M.A. Frumkin, Automatic Recognition of Performance Idioms in Scientific Applications, Parallel & Distributed Processing Symposium (IPDPS), 2011, pp.118-127.

[E.Ipek 2005] E. Ipek, B. R. de Supinski, M. Schulz, and S. A. McKee, “An Approach to Performance Prediction for Parallel Applications,” in Euro-Par, Aug 2005,pp. 196–205.

[E.Ipek 2006] E. `Ipek, S. McKee, B. de Supinski, M. Schulz, and R. Caruana. Efficiently exploring architectural design spaces via predictive modeling. In Architectural Support for Programming Languages and Operating Systems (ASPLOS XII), October 2006.

[S.H. Jang, 2005] Seung-Hye Jang, Valerie Taylor, Xingfu Wu, Mieke Prajugo, Ewa Deelman, Gaurang Mehta, Karan Vahi, Performance Prediction-based versus Load-based Site Selection: Quantifying the Difference, the 18th International Conference on Parallel and Distributed Computing Systems (PDCS-2005), Las Vegas, Nevada, 12 -14 September 2005.

[S.A. Jarvis 2006] Stephen A. Jarvis, Daniel P. Spooner, Helene N. Lim Choi Keung, Junwei Cao, Subhash Saini, and Graham R. Nudd. 2006. Performance prediction and its use in parallel and distributed computing systems. Future Gener. Comput. Syst. 22, 7 (August 2006), 745-754.

[N.K.Kapoor 2010] Kapoor, N.K., Majumdar, S., Nandy, B., “Class Based Grid Resource Management Strategies for On Demand Jobs”, Simulation: Transactions of the Society for Modeling and Simulation International Vol. 86, No.: 11, November 2010, pp. 675-697.

[D.J. Kerbyson 2002] Darren J. Kerbyson, Harvey J. Wasserman, Adolfy Hoisie. Performance Prediction - A tool for Advanced Architecture Exploration. In Proceedings of the Int. Workshop on Innovative Architecture, Big Island, HI., January 2002.

[D.J.Kerbyson 2005] Darren J. Kerbyson, Philip W. Jones. A Performance Model of the Parallel Ocean Program. In Int. J. High Performance Computing Applications, 19 (5): 261--276, August 2005. LA-UR 04-8793

[S.Krishnaswamy 2004] Krishnaswamy, S.; Loke, S.W.; Zaslavsky, A.; Monash Univ., Clayton, Vic., Australia, Estimating computation times of data-intensive applications, Distributed Systems Online, IEEE, Volume 5, Issue 4, April 2004.

[K. Kurowski 2005] Krzysztof Kurowski, Ariel Oleksiak, Jarek Nabrzyski, Agnieszka Kwiecień, Marcin Wojtkiewicz, Maciej Dyczkowski, Francesc Guim, Julita Corbalan and Jesus Labarta, Multicriteria Grid Resource Management Using Performance Prediction Techniques, CoreGrid Integration Workshop, Pisa, Italy, November 2005, Springer CoreGRID Proceedings, vol. 4.

[B.Lafreniere 2005] B. Lafreniere and A.C. Sodan. ScoPred—Scalable User-Directed Performance Prediction Using Complexity Modeling and Historical Data. Proc. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Cambridge, LNCS 3834, Springer, June 2005.

[C.B.Lee 2004] C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely, “Are user runtime estimates inherently inaccurate? ”. In 10th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), pp. 253–263, Springer-Verlag, Jun 2004. Lect. Notes Comput. Sci. vol. 3277.

[B.C. Lee 2007] B. C. Lee, D. M. Brooks, B. R. de Supinski, M. Schulz, K. Singh, and S. A. McKee, “Methods of Inference and Learning for Performance Modeling of Parallel Applications,” in PPOPP, 2007, pp. 249–258.

[J.L. Lerida, 2008] Joseph Ll. Lérida , F. Solsona , F. Giné , J. R. García , M. Hanzich , P. Hernández, Enhancing Prediction on Non-dedicated Clusters, Proceedings of the 14th international Euro-Par conference on Parallel Processing, August 26-29, 2008, Las Palmas de Gran Canaria, Spain

[K.Levenberg 1944] K. Levenberg. A Method for the Solution of Certain Problems in Least Squares. Quart. Appl. Math., Vol. 2, 1944, pp. 164–168.

[H.Li 2005] H. Li, D. Groep, and L.Wolters. Efficient response time prediction by exploiting application and resource state similarities. In proceedings of 4th IEEE/ACM International Workshop on Grid Computing (Grid'05), 2005.

[J.Li 2009] J. Li, Xiaosong Ma, K. Singh, M. Schulz, B.R. de Supinski, S.A. McKee, Machine learning based online performance prediction for runtime parallelization and task scheduling, in IEEE International Symposium on Performance Analysis of Systems and Software, 2009, pp. 89-100.

[U.Liblin 2003] U. Lublin and D. G. Feitelson, “The workload on parallel supercomputers: modeling the characteristics of rigid jobs”. J. Parallel & Distributed Comput. 63(11), pp. 1105-1122, Nov 2003.

[M.I.A.Lourakis 2005] M.I.A. Lourakis. levmar: Levenberg-Marquardt nonlinear least squares algorithms in C/C++. http://www.ics.forth.gr/~lourakis/levmar/, retrieved Jan 2005.

[G.Marin 2004] G. Marin and J. Mellor-Crummey. Cross-Architecture Predictions for Scientific Applications Using Parameterized Models. Proc. SIGMETRICS, New York, NY, USA, June 2004.

[G.Marin 2007] G. Marin and J. Mellor-Crummey. Application insight through performance modeling. In IEEE International Performance Computing and Communications Conference, Apr 2007.

[M.M.Mathis 2005] Mark M. Mathis, Darren J. Kerbyson. A General Performance Model of Structured and Unstructured Mesh Particle Transport Computations. In J. Supercomputing, 34 (2): 181--199, November 2005. LA-UR 04-8794

[M.M.Mathis 2006] Mark M. Mathis, Darren J. Kerbyson. Dynamic Performance Prediction of an Adaptive Mesh Application. In Proceedings of the Workshop on System

Management Tools for Large-Scale Parallel Systems, IEEE/ACM Int. Parallel and Distibuted Processing Symposium (IPDPS), Rhodes, Greece, April 2006.

[A.Matsunaga 2010] Andrea Matsunaga and Jose A. B. Fortes, 2010. On the Use of Machine Learning to Predict the Time and Resources Consumed by Applications. In Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID '10). IEEE Computer Society, Washington, DC, USA, 495-504.

[T.N.Minh 2010] T.N. Minh, and L. Wolters, Using Historical Data to Predict Application Runtimes on Backfilling Parallel Systems, 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, 2010, pp. 246-252

[A.W.Mu’alem 2001] A. W. Mu'alem, and D. G. Feitelson, “Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling,” IEEE Trans. Parallel Distrib. Syst., vol. 12, no. 6, 2001, pp. 529-543.

[F.Nadeem 2006] Farrukh Nadeem, Muhammad Murtaza Yousaf, Radu Prodan, and Thomas Fahringer. 2006. Soft Benchmarks-Based Application Performance Prediction Using a Minimum Training Set. In Proceedings of the Second IEEE International Conference on e-Science and Grid Computing (E-SCIENCE '06). IEEE Computer Society, Washington, DC, USA, pp. 71-.

[F.Nadeem 2009] Farrukh Nadeem and Thomas Fahringer. 2009. Using Templates to Predict Execution Time of Scientific Workflow Applications in the Grid. In Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09). IEEE Computer Society, Washington, DC, USA, 2009, pp. 316-323.

[V.K.Naik 1997] V.K. Naik, S.K. Setia, and M.S. Squillante. Processor Allocation in Multiprogrammed Distributed-Memory Parallel Computer Systems. J. of Parallel and Distr. Computing, 46(1), 1997, pp. 28-47.

[M.Nakazawa 2005] Mario Nakazawa, David K. Lowenthal, and Wendou Zhou. 2005. The MHETA Execution Model for Heterogeneous Clusters. In Proceedings of the 2005 ACM/IEEE conference on Supercomputing (SC '05). IEEE Computer Society, Washington, DC, USA, 7-.

[Nirav 1999] Nirav H. Kapadia, J.A.B. Fortes, and Carla E. Brodley. 1999. Predictive Application-Performance Modeling in a Computational Grid Environment. In Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC '99). IEEE Computer Society, Washington, DC, USA, 6-.

[G.R.Nudd 2000] G.R. Nudd, D.J. Kerbyson, E.Papaefstathiou, J.S. Harper, S.C. Perry and D.V. Wilcox. PACE: A Toolset for the Performance Prediction of Parallel and Distributed Systems. The International Journal of High Performance Computing, 4:228–251, 2000.

[OpenMP 2008] OpenMP Official Web Site at http://openmp.org/wp/, retrieved June 2008.

[Parallel Workload Archive] Parallel Workload Archive, http://www.cs.huji.ac.il/labs/parallel/workload/, Retrieved July 2011.

[E. Perelman 2006] Erez Perelman, Marzia Polito, Jean-Yves Bouguet, John Sampson, Brad Calder, and Carole Dulong. 2006. Detecting phases in parallel applications on shared memory architectures. In Proceedings of the 20th international conference on Parallel and distributed processing (IPDPS'06). IEEE Computer Society, Washington, DC, USA, 88-88.

[D.Perkovic 2001] D. Perkovic and P.J. Keleher, "Randomization, Speculation, and Adaptation in Batch Schedulers," Supercomputing, p. 7, Sept. 2000.

[A.Petitet] A.Petitet, R.Whaley, J.Dongarra, and A.Cleary. HPL - A portable implementation of the high-performance LINPACK benchmark for distributed-memory computers. www.netlib.org/benchmark/hpl.

[W.Pfeiffer 2008] Wayne Pfeiffer and Nicholas J. Wright, Modeling and Predicting Application Performance on Parallel Computers Using HPC Challenge Benchmarks, 22nd IEEE International Parallel and Distributed Processing Symposium, April 2008

[S.Pllana 2005] Sabri Pllana, Thomas Fahringer: Performance Prophet: A Performance Modeling and Prediction Tool for Parallel and Distributed Programs. ICPP Workshops 2005: 509-516

[POP Application] “Parallel Ocean Program”: http://climate.lanl.gov/Models/POP/

[G.Rodriguez 2004] G. Rodr´ıguez, R. M. Badia, J. Labarta ”Generation of Simple Analytical Models for Message Passing Applications” Proceedings of European Conference on Parallel Processing Euro-Par 2004: p183-188

[J.C.Sancho 2006] Jose C Sancho, Kevin J Barker, Darren J Kerbyson, Kei Davis. Quantifying the Potential Benefit of Overlapping Communication and Computation in Large-Scale Scientific Applications. In Proceedings of the IEEE/ACM Conference on Supercomputing (SC'06), Tampa, FL, November 2006. LA-UR 06-3109

[H.A.Sanjay 2008] H. A. Sanjay , Sathish Vadhiyar, Performance modeling of parallel applications for grid scheduling, Journal of Parallel and Distributed Computing, v.68 n.8, p.1135-1145, August, 2008

[R.Sarikaya 2010] Ruhi Sarikaya, Canturk Isci, and Alper Buyuktosunoglu. 2010. Program behavior prediction using a statistical metric model. In Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems (SIGMETRICS '10). ACM, New York, NY, USA, 371-372.

[J. Schopf 1998] Jennifer Schopf and Francine Berman. Performance Prediction in Production Environments. In 14th International Parallel Processing Symposium and the 9th Symposium on Parallel and Distributed Processing, 1998.

[SDSC95] SDSC Par95 log files from the parallel workload archive, http://www.cs.huji.ac.il/labs/parallel/workload/l_sdsc_par/index.html, retrieved July 2011.

[SDSC96] SDSC Par96 log files from the parallel workload archive, http://www.cs.huji.ac.il/labs/parallel/workload/l_sdsc_par/index.html, retrieved July 2011.

[SHARCNET 2009] SHARCNET project. http://www.sharcnet.ca, retrieved June 2009.

[S.Sharkawi 2009] Sameh Sharkawi, Don DeSota, Raj Panda, Rajeev Indukuru, Stephen Stevens, Valerie Taylor, and Xingfu Wu. 2009. Performance projection of HPC applications using

SPEC CFP2006 benchmarks. In Proceedings of the 2009 IEEE International Symposium on Parallel &Distributed Processing (IPDPS '09). IEEE Computer Society, Washington, DC, USA, 1-12.

[S.Shimizu 2009] Shuichi Shimizu, Raju Rangaswami, Hector A. Duran-Limon, and Manuel Corona-Perez. 2009. Platform-independent modeling and prediction of application resource usage characteristics. J. Syst. Softw. 82, 12 (December 2009), 2117-2127.

[E.Shmueli 2006] E. Shmueli and D. G. Feitelson, "Using site-level modeling to evaluate the performance of parallel system schedulers," in MASCOTS '06: Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation. Washington, DC, USA: IEEE Computer Society, 2006, pp. 167–178.

[E.Shmueli, 2007] E. Shmueli and D. G. Feitelson, "Uncovering the effect of system performance on user behavior from traces of parallel systems". In 15th Conf. Modeling, Analysis, & Simulation of Comput. & Telecomm. Syst., pp. 274-280, Oct 2007.

[E.Shmueli 2009] Edi Shmueli and Dror G. Feitelson. 2009. On Simulation and Design of Parallel-Systems Schedulers: Are We Doing the Right Thing?. IEEE Trans. Parallel Distrib. Syst. 20, 7 (July 2009), 983-996.

[K.Singh 2010] Karan Singh, Matthew Curtis-Maury, Sally A. McKee, Filip Blagojevi\&\#263;, Dimitrios S. Nikolopoulos, Bronis R. de Supinski, and Martin Schulz. 2010. Comparing scalability prediction strategies on an SMP of CMPs. In Proceedings of the 16th international Euro-Par conference on Parallel processing: Part I (EuroPar'10), Pasqua D'Ambra, Mario Guarracino, and Domenico Talia (Eds.). Springer-Verlag, Berlin, Heidelberg, 143-155.

[W.Smith 1998] W. Smith, I. Foster, and V. Taylor, Predicting Application Run Times Using Historical Information, In Proceedings of the IPPS/SPDP '98Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

[W.Smith 1999] W. Smith, V. E. Taylor, and I. T. Foster. Using run-time predictions to estimate queue wait times and improve scheduler performance. Proceedings of the Job

Scheduling Strategies for Parallel Processing, Lecture Notes In Computer Science; Vol. 1659:202 – 219, 1999.

[W.Smith 2004] W. Smith, I. Foster, and V. Taylor. Predicting Application Run Times With Historical Information. Journal of Parallel and Distributed Computing, 64 (9), 2004, pp. 1007-1016.

[W.Smith 2007] W. Smith. Prediction Services for Distributed Computing. In Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium, April 2007, pp. 1-10.

[W.Smith 2010] W. Smith, A Service for Queue Prediction and Job Statistics, Gateway Computing Environments Workshop (GCE), 2010, Issue 14-14 Nov. 2010, pp. 1-8. USA.]

[A.Snavely 2001] A. Snavely, L. Carrington, and N. Wolter. Modeling Application Performance by Convolving Machine Signatures with Application Profiles. Proc. IEEE Ann. Workshop on Workload Characterization, 2001.

[A.Snavely 2003] A. Snavely, X. Gao, C. Lee, N. Wolter, & J. Labarta, "Performance modeling of HPC applications", Parallel Computing, Dresden, 2003.

[A.C.Sodan 2009] A.C. Sodan. Adaptive Scheduling for QoS Virtual Machines under Different Resource Availability - Performance Effects and Predictability. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP) of IPDPS, Springer, May 2009.

[A.C.Sodan 2006] A.C. Sodan and X. Huang. Adaptive Time/Space Scheduling with SCOJO. International Journal of High Performance Computing and Networking (IJHPCN), Vol. 4, Nos. 5/6, 2006.

[S.Sodhi, 2008] Sukhdeep Sodhi, Jaspal Subhlok, and Qiang Xu. 2008. Performance prediction with skeletons. Cluster Computing 11, 2 (June 2008), 151-165.

[X.H.Sun 1999] X.H. Sun, M. Pantano, T. Fahringer, and Z. Zhan. SCALA: A Framework for Performance Evaluation of Scalable Computing. Proc. IPPS/SPDP Workshops, 1999.

[D. Talby, 2006] Session-based, estimation-less, and information-less runtime prediction algorithms for parallel and grid job scheduling, David Talby, Dan Tsafrir, Zviki Goldberg, Dror G. Feitelson, Technical Report 2006-77, School of Computer Science and Engineering, the Hebrew University, August 2006.

[V. Taylor 2001] V. Taylor, X. Wu, J. Geisler, X. Li, z. Lan, M. Hereld, I. Judson, and R. Stevens. Prophesy: Automating the modeling process. In Proc. Of the Third International Workshop on Active Middleware Services, 2001.

[V. Taylor 2002] Valerie Taylor, Xingfu Wu, Jonathan Geisler, and Rick Stevens. 2002. Using Kernel Couplings to Predict Parallel Application Performance. In Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC '02). IEEE Computer Society, 125-.

[V. Taylor 2003] Valerie Taylor, Xingfu Wu, and Rick Stevens. 2003. Prophesy: an infrastructure for performance analysis and modeling of parallel and grid applications. SIGMETRICS Perform. Eval. Rev. 30, 4 (March 2003), 13-18.

[TeraGrid] The TeraGrid infrastructure, https://www.teragrid.org, retrieved January 2011.

[D.Thomas 2010] D. Thomas, J.P. Panziera, and J. Baron, MPInside: a performance analysis and diagnostic tool for MPI applications, In Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering, 2010, pp. 79-86.

[M.Tikir 2009] Mustafa M. Tikir, Michael A. Laurenzano, Laura Carrington, and Allan Snavely. 2009. PSINS: An Open Source Event Tracer and Execution Simulator for MPI Applications. In Proceedings of the 15th International Euro-Par Conference on Parallel Processing (Euro-Par '09), Henk Sips, Dick Epema, and Hai-Xiang Lin (Eds.). Springer-Verlag, 135-148.

[A.Tiwari 2009] Ananta Tiwari, Vahid Tabatabaee, and Jeffrey K. Hollingsworth. 2009. Tuning parallel applications in parallel. Parallel Comput. 35, 8-9 (August 2009), 475-492.

[A.Toomula 2004] Toomula, A., Subhlok, J.: Replication memory behavior for performance prediction. In: LCR 2004: The 7th Workshop on Languages, Compilers, and Run-time Support for Scalable Systems, Houston, TX, October 2004

[D.Tsafrir, 2006] D. Tsafrir, D.G. Feitelson, "Instability in parallel job scheduling simulation: the role of workload flurries," Parallel and Distributed Processing Symposium, International, p. 54, Proceedings 20th IEEE International Parallel & Distributed Processing Symposium, 2006

[D.Tsafrir 2007] Dan Tsafrir, Yoav Etsion, and Dror G. Feitelson. 2007. Backfilling Using System-Generated Predictions Rather than User Runtime Estimates. IEEE Trans. Parallel Distrib. Syst. 18, 6 (June 2007), 789-803.

[D.Tsafrir, 2010] Dan Tsafrir. 2010. Using inaccurate estimates accurately. In Proceedings of the 15th international conference on Job scheduling strategies for parallel processing (JSSPP'10), Eitan Frachtenberg and Uwe Schwiegelshohn (Eds.). Springer-Verlag, Berlin, Heidelberg, 208-221.

[B.M.Tudor 2011] B. Tudor and Y.M. Teo, A Practical Approach for Performance Analysis of Shared Memory Programs, Proceedings of 25th IEEE International Parallel & Distributed Processing Symposium (IPDPS), 2011, pp. 652-663.

[Valgrind] Valgrind debugging tool, http://valgrind.org/, retrieved on July 2011.

[S. Venkataramaiah 2003] Venkataramaiah, S., Subhlok, J.: Performance estimation for scheduling on shared networks. In: 9thWorkshop on Job Scheduling Strategies for Parallel Processing, Seattle, WA, June 2003.

[M.Wall 2009] M. Wall, GAlib: A C++ Library of Genetic Algorithm Components. MIT, http://lancet.mit.edu/ga/, retrieved July 2009.

[Z.Wang 2009] Zheng Wang and Michael F.P. O'Boyle. 2009. Mapping parallelism to multi-cores: a machine learning based approach. In Proceedings of the 14th ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP '09). ACM, 75-84.

[X.Wu 2004] Xingfu Wu, Valerie Taylor, Jonathan Geisler, and Rick Stevens, Isocoupling: Reusing Coupling Values to Predict Parallel Application Performance, the 18th International Parallel and Distributed Processing Symposium (IPDPS2004), Santa Fe, New Mexico, April 26-30, 2004.

[X. Wu 2006_1] Xingfu Wu, Valerie Taylor, and Joseph Paris, A Web-based Prophesy Automated Performance Modeling System, the International Conference on Web Technologies, Applications and Services (WTAS2006), July 17-19, 2006.

[X. Wu 2006_2] Xingfu Wu, Valerie Taylor, Shane Garrick, Dazhi Yu, and Jacques Richard, Performance Analysis, Modeling and Prediction of a Parallel Multiblock Lattice Boltzmann Application Using Prophesy System, IEEE International Conference on Cluster Computing, September 25-28, 2006, Barcelona, Spain.

[X. Wu 2009] Wu, X., Taylor, V.: Using processor partitioning to evaluate the performance of MPI, OpenMP and hybrid parallel applications on dual- and quad-core Cray XT4 systems. In: Cray User Group Conference, May 4-7 (2009).

[R.Wu 2008] Rongteng Wu , Jizhou Sun , Jinyan Chen, Parallel execution time prediction of the multitask parallel programs, Performance Evaluation, v.65 n.10, p.701-713, October, 2008

[Q.Xu 2008] Qiang Xu and Jaspal Subhlok. 2008. Construction and evaluation of coordinated performance skeletons. In Proceedings of the 15th international conference on High performance computing (HiPC'08), Ponnuswamy Sadayappan, Manish Parashar, Ramamurthy Badrinath, and Viktor K. Prasanna (Eds.). Springer-Verlag, Berlin, Heidelberg, 73-86.

[L.T.Yang 2005] L. T. Yang, X. Ma, F. Mueller, ”Cross-Platform Performance Prediction of Parallel Applications Using Partial Execution,” sc, p. 40, ACM/IEEE SC 2005 Conference (SC’05), 2005

[J.Zhai 2010] Jidong Zhai, Wenguang Chen, and Weimin Zheng. 2010. PHANTOM: predicting performance of parallel applications on large-scale parallel machines using a single node. SIGPLAN Not. 45, 5 (January 2010), pp. 305-314.

[Zilber 2005] Zilber, O. Amit, and D. Talby. What is worth learning from parallel workloads? a user and session based analysis. In Proc. 19th intl. conf. Supercomputing, pages 377–386, Jun 2005.

[D.Zotkin 1999] D. Zotkin and P.J. Keleher, "Job-Length Estimation and Performance in Backfilling Schedulers," Proc. IEEE Int'l Symp. High Performance Distributed Computing (HPDC), p. 39, Aug. 1999.

APPENDICES

Appendix A. Envelope Derivation via Closed-form Solution Formula[1]

As specified in chapter 3, ADEPT depends on a closed-form solution to derive an envelope in order to constrain the search space. For the sake of clarity, we only briefly discussed the closed-form solution and provided a few examples of the formula in Section 3.5. This appendix provides the details on derivation of all the closed-form solution formulas, and on how these formulas are used by ADEPT to derive the envelope.

Closed-form Solution Formulas

The closed-form solution is derived for pairs of observations $< n_i, t_i >, < n_j, t_j >$ of an instance of the Downey model. The solution separately addresses high variance and low variance model instances. We make the following assumptions:

1. Runtimes are greater than one second.
2. There exist at least three observations. The third observation will be referred to as $< n_k, t_k >$.
3. Not all three t_i, t_j, and t_k values are equal.
4. $n_i < n_j$.

[1] This Appendix was published as a technical report at the University of Windsor: [A.Deshmeh 2009]

We first discuss cases where observations are drawn from a high variance model instances.

Case 1: Assuming the observations are both placed in the first piece of the runtime function results in the following equations:

$$t_i = \sigma + \frac{A + A\sigma - \sigma}{n_i} \tag{5}$$

$$t_j = \sigma + \frac{A + A\sigma - \sigma}{n_j} \tag{6}$$

Solving the above system of equations for *A* and *σ* results in the following equations:

$$\sigma = \frac{n_j t_j - n_i t_i}{n_j - n_i} \tag{7}$$

$$A = \frac{n_i t_i - \sigma(n_i - 1)}{\sigma + 1} \tag{8}$$

Case 2: Assuming that observation $< n_i, t_i >$ is placed in the first piece and observation $< n_j, t_j >$ is placed in the second piece of runtime function results in the following equations. Note that due to Assumption 4 and definition of runtime function (see Section 3) the reverse order is not possible.

$$t_i = \sigma + \frac{A + A\sigma - \sigma}{n_i} \tag{9}$$

$$t_j = \sigma + 1 \tag{10}$$

Solving the above equation system for *A* and *σ* results in the following equations:

$$\sigma = t_j - 1 \tag{11}$$

$$A = \frac{\sigma(1-n_i)+n_i t_i}{\sigma+1} \tag{12}$$

Case 3: Assuming two observations are placed on the second piece of the runtime function will result in the following equations:

$$t_i = \sigma + 1 \tag{13}$$

$$t_j = \sigma + 1 \tag{14}$$

Here, the observations will only provide the value of *σ*. However, according to Assumption 2, there exists a third observation $< n_k, t_k >$. This observation has to be in the first piece of the runtime function, as otherwise all observations will have the same runtimes, contradicting Assumption 3. This means that Case 3 results in an equation system similar to Case 2, with observation $< n_k, t_k >$ in the first piece of the runtime function, and observation $< n_j, t_j >$ in the second piece. *A* and *σ* therefore are calculated as:

$$\sigma = t_j - 1 \tag{15}$$

$$A = \frac{\sigma(1-n_k)+n_k t_k}{\sigma+1} \tag{16}$$

For observations drawn from a low variance instance of the Downey model, Assumptions 1, 2, and 3, combined with the definition of runtime function, will guarantee that either two of the observations are in the first piece, or two of the observations are in the second piece of the runtime function. Therefore, for obtaining the underlying model instance it is sufficient to consider these two cases.

Case 4: This case assumes that both observations are placed on the first piece of the runtime function, resulting in the following equations:

$$t_i = \frac{A - \sigma/2}{n_i} + \sigma/2 \quad (17)$$

$$t_j = \frac{A - \sigma/2}{n_j} + \sigma/2 \quad (18)$$

Solving the above equation system for *A* and *σ* results in the following equations (note that these were already shown in Section 4.5)

$$\sigma = \frac{2(n_j t_j - n_i t_i)}{n_j - n_i} \quad (19)$$

$$A = n_i t_i - \sigma(n_i - 1)/2 \quad (20)$$

Case 5: Assuming that both observations are placed in the second piece of the runtime function results in the following equations:

$$t_i = \sigma \frac{A - 1/2}{n_i} + 1 - \sigma/2 \quad (21)$$

$$t_j = \sigma \frac{A - 1/2}{n_j} + 1 - \sigma/2 \quad (22)$$

Solving the above equation system for *A* and *σ* results in the following equations:

$$\sigma = \frac{2(n_i t_i - n_j t_j)}{n_j - n_i} + 2 \quad (23)$$

$$A = \frac{n_i(t_i - 1)}{\sigma} + \frac{n_i + 1}{2} \tag{24}$$

Although more than one instance might be obtained per observation pair, corresponding to low variance and high variance modes, these can be reduced to one either due to contradiction (e.g. $\sigma > 1$ for low variance), or by choosing instances that match all observation pairs.

Note that (7), (8), (11), (12), (15), (16), (19), (20), (23), and (24) are the final formulas mentioned in Section 3.5.1.

Envelope Derivation Formulas

To derive envelope formulas, as noted in Section3.5.1, we assume that each observation $< n_i , t_i >$deviates from the underlying model by at most δ up or down. Thus, if the runtime value produced by the underlying model at n_i nodes ist_i', the following results:

$$t_i' \in t_i * [1 - \delta, 1 + \delta] \tag{25}$$

Closed-form solutions should be calculated using runtime values produced by the underlying model. Since these values are not available, the range in which it falls has to be used instead, as obtained from Relation (27). For this purpose, the closed-form solution formulas are extended to envelope formulas, which calculate ranges instead of exact values for the underlying model's parameters. All the five Cases 1, 2, 3, 4, and 5 from the closed-form solution above are extended in the following to incorporate δ and produce ranges.

For Case 1, Equations (7) and (8) can be extended to:

$$\sigma \in \frac{n_j t_j - n_i t_i}{n_j - n_i} [1 - \delta, 1 + \delta] \tag{26}$$

$$A \in \frac{1}{\sigma+1}\left[n_i t_i (1-\delta) - \sigma(n_i - 1)\,,\, n_i t_i (1+\delta) - \sigma(n_i - 1)\right] \tag{27}$$

For Case 2, Equations (11) and (12) can be extended to:

$$\sigma \in \left[t_i (1-\delta) - 1\,,\, t_i (1+\delta) - 1\right] \tag{28}$$

$$A \in \frac{1}{\sigma+1}\left[n_i t_i (1-\delta) + \sigma(1 - n_i)\,,\, n_i t_i (1+\delta) + \sigma(1 - n_i)\right] \tag{29}$$

For Case 3, Equations (15) and (16) can be extended to:

$$\sigma \in \left[t_j (1-\delta) - 1\,,\, t_j (1+\delta) - 1\right] \tag{30}$$

$$A \in \frac{1}{\sigma+1}\left[n_k t_k (1-\delta) + \sigma(1 - n_k)\,,\, n_k t_k (1+\delta) + \sigma(1 - n_k)\right] \tag{31}$$

Note that (31) is essentially the same as (29), calculated using a different observation.

For Case 4, Equations (19) and (20) can be extended to produce the following formulas (note that these were already shown in Section 4.5)

$$\sigma \in \frac{2(n_j t_j - n_i t_i)}{n_j - n_i}\left[1-\delta\,,\, 1+\delta\right] \tag{32}$$

$$A \in \left[(n_i t_i)(1-\delta) - \sigma(n_i - 1)/2, (n_i t_i)(1+\delta) - \sigma(n_i - 1)/2\right] \tag{33}$$

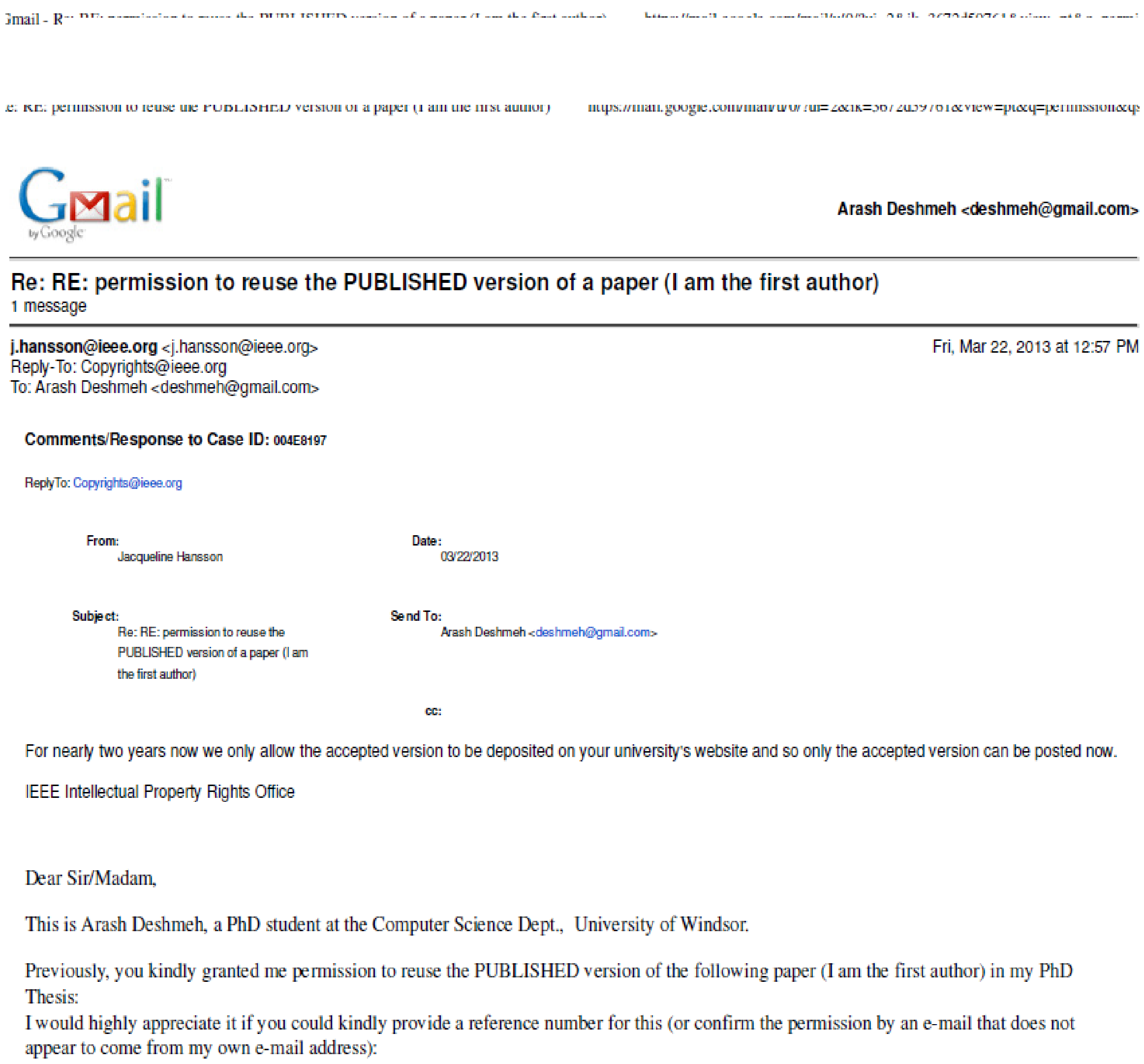
For Case 5, Equations (23) and (24) can be extended to:

$$\sigma \in \left[\frac{2(n_i t_i - n_j t_j)}{n_j - n_i}(1-\delta) + 2\,,\, \frac{2(n_i t_i - n_j t_j)}{n_j - n_i}(1+\delta) + 2\right] \tag{34}$$

$$A \in \left[(n_i + 1)/2 - n_i/\sigma + n_i t_i (1-\delta)/\sigma, (n_i + 1)/2 - n_i/\sigma + n_i t_i (1+\delta)/\sigma\right] \tag{35}$$

It should be noted that for each observation pair, one of the five cases holds, and therefore the underlying model's parameters are guaranteed to be in the ranges calculated using the corresponding formula. The formula for the other cases will then only add to the search space and will not affect the solution. Also, for σ ranges, parts of the range which fall below 0 for all cases or above one for Cases 4 and 5 are discarded as these values would be invalid for σ. The same applies to parts of A ranges that fall below 1.

Appendix B. Permission letter to include previous publication



Gmail - R [illegible]

e: RE: permission to reuse the PUBLISHED version of a paper (I am the first author)   https://mail.google.com/mail/u/0/?ui=2&ik=3672d59761&view=pt&q=permission&q

Arash Deshmeh <deshmeh@gmail.com>

Re: RE: permission to reuse the PUBLISHED version of a paper (I am the first author)
1 message

j.hansson@ieee.org <j.hansson@ieee.org>   Fri, Mar 22, 2013 at 12:57 PM
Reply-To: Copyrights@ieee.org
To: Arash Deshmeh <deshmeh@gmail.com>

Comments/Response to Case ID: 004E8197

ReplyTo: Copyrights@ieee.org

From: Jacqueline Hansson	**Date:** 03/22/2013
Subject: Re: RE: permission to reuse the PUBLISHED version of a paper (I am the first author)	**Send To:** Arash Deshmeh <deshmeh@gmail.com>
	cc:

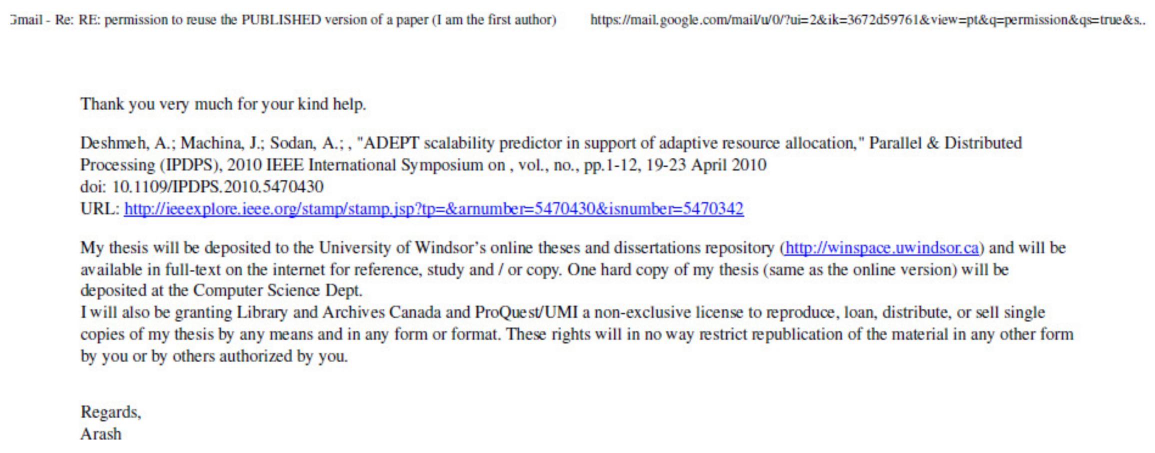
For nearly two years now we only allow the accepted version to be deposited on your university's website and so only the accepted version can be posted now.

IEEE Intellectual Property Rights Office

Dear Sir/Madam,

This is Arash Deshmeh, a PhD student at the Computer Science Dept., University of Windsor.

Previously, you kindly granted me permission to reuse the PUBLISHED version of the following paper (I am the first author) in my PhD Thesis:
I would highly appreciate it if you could kindly provide a reference number for this (or confirm the permission by an e-mail that does not appear to come from my own e-mail address):



Thank you very much for your kind help.

Deshmeh, A.; Machina, J.; Sodan, A.; , "ADEPT scalability predictor in support of adaptive resource allocation," Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on , vol., no., pp.1-12, 19-23 April 2010
doi: 10.1109/IPDPS.2010.5470430
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5470430&isnumber=5470342

My thesis will be deposited to the University of Windsor's online theses and dissertations repository (http://winspace.uwindsor.ca) and will be available in full-text on the internet for reference, study and / or copy. One hard copy of my thesis (same as the online version) will be deposited at the Computer Science Dept.
I will also be granting Library and Archives Canada and ProQuest/UMI a non-exclusive license to reproduce, loan, distribute, or sell single copies of my thesis by any means and in any form or format. These rights will in no way restrict republication of the material in any other form by you or by others authorized by you.

Regards,
Arash

VITA AUCTORIS

NAME:	Gholamhossein Deshmeh
PLACE OF BIRTH:	Mashad
YEAR OF BIRTH:	1981
EDUCATION:	Alborz High School, Tehra, 1999
	Tehran Polytechnic University, B.Sc., Tehran, 2004
	Tehran Polytechnic University, M.Sc., Tehran, 2007
	University of Windsor, Ph.D., Windsor, ON, 2013