# ADEPT Scalability Predictor in Support of Adaptive Resource Allocation

Arash Deshmeh
School of Computer Science
University of Windsor
Windsor, Canada
deshmeh@uwindsor.ca

Jacob Machina
School of Computer Science
University of Windsor
Windsor, Canada
machina@uwindsor.ca

Angela Sodan
School of Computer Science
University of Windsor
Windsor, Canada
acsodan@uwindsor.ca

***Abstract*—Adaptive resource allocation with different numbers of machine nodes provides more flexibility and significantly better potential performance for local job and grid scheduling. With the emergence of parallel computing in every-day life on multi-core systems, such schedulers will likely increase in practical relevance. A major reason why adaptive schedulers are not yet practically used is lacking knowledge of the scalability curves of the applications. Existing white-box approaches for scalability prediction are too expensive to apply them routinely. We present ADEPT, a speedup and runtime prediction tool, which is inexpensive and easy-to-use. ADEPT employs a black-box model and can be practically applied at large scale without user or administrator involvement. ADEPT requires neither program analysis and measurements nor user guesses but makes highly accurate predictions with only few observations of application runtime over different numbers of nodes/cores. ADEPT performs efficient model fitting by introducing an envelope-derivation technique to constrain the search. Additionally, ADEPT is capable of handling deviations from the underlying model by detection and automatic correction of anomalies via a fluctuation metric and by considering specific scalability patterns via multi-phase modeling. ADEPT also performs reliability judgment with potential proposal for placement of additional observations. Using MPI and OpenMP implementations of the NAS benchmarks and seven real applications, we demonstrate the effectiveness and high prediction accuracy of ADEPT for both speedup and runtime prediction, including interpolative and extrapolative cases, and show the capability of ADEPT to successfully handle special cases.**

***Keywords-component; job schedulers; adaptive resource allocation; performance prediction; scalability; black-box model***

## I. INTRODUCTION

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 [1][2][3][4]. 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 [1][5]. 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 [2]. 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 [6]. 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. [7]) as feedback to users and system administrators. However, such scalability testing in supercomputing centers lacks a sound model. Though so far mostly applied 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 [5]. Luckily, OpenMP applications on multi-core SMP servers were found to exhibit similar shapes of speedup/runtime curves as MPI applications on clusters [8]. 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. [9][10][11][7]. 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

978-1-4244-6443-2/10/$26.00 ©2010 IEEE

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 [12]. 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 superlinear speedups, and due to reliability of a specific prediction being hard to judge.

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 superlinear speedup.

Experiments with the NAS benchmarks [13] 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.

## II. Related Work

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 [14] 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 [15] 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 [10] applies multiple linear regression 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 [12]. To validate the proposed model, the NAS benchmark suite [1] 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 [16] threads is proposed in [17]. 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 [11] 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 [18] 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 [9], which models both the application and the architecture based on their "fundamental operations" capability. The SCALA system [7] 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 [19], which employs regression to predict scalability. The approach proposed in [20] constructs separate models for computation and communication, and uses them as input to a simulator for runtime prediction.

## III. The Downey Model

### A. Overview

Downey proposed a black-box model which describes an application via two parameters: $A$ as the maximum parallelism and $\sigma$ which represents how fast the maximum is reached, i.e. describes the shape of the curve [12]. 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.

TABLE 1. S AND T PIECEWISE FUNCTIONS OF DOWNEY MODEL.

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

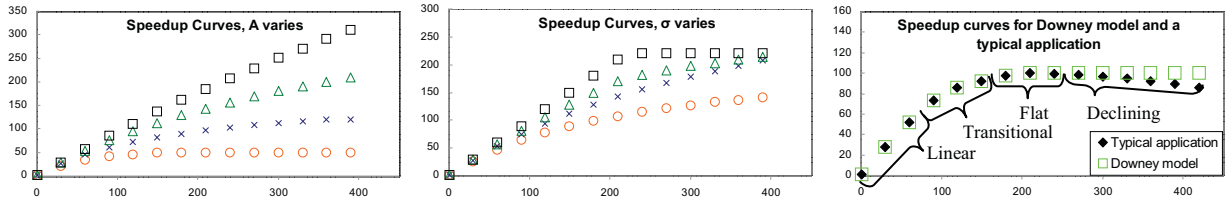


Figure 1. (a) Speedup curve: σ=2, varying A (1000, 300, 120, 50), (b) Speedup curve: A=220, varying σ (0, 0.5, 1, 1000), (c) Downey model's lack of support for declining piece of the speedup curve. Graphs show S over N.

In Table 1, n represents the number of nodes, T(n) and S(n) represent the runtime and speedup on n nodes. Figure 1 (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.

### B. *Strengths and Weaknesses*

The Downey model benefits mainly from the fact that it uses only two parameters (namely, A and $\sigma$). This makes the model easier to store and understand, and reduces the number of observations necessary to learn the parameters for a 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 1(c). This is insignificant as the maximum meaningful number of nodes can be obtained as $2A-1$ for low variance mode and as $A+A\sigma-\sigma$ for high variance mode. Also, the processor working set—proposed as a metric to determine a balance between speedup and resource consumption [21]—could be calculated using the fitted model, by finding the minimum $n$ such that $\eta(n)$ is maximal, with $\eta(n)=S^2(n)/n$.

## IV. THE ADEPT PREDICTOR

ADEPT needs to learn $A$ and $\sigma$ 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. To address these challenges and provide an efficient predictor which is applicable in production environments, ADEPT is composed of four major components (see Figure 2):

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.

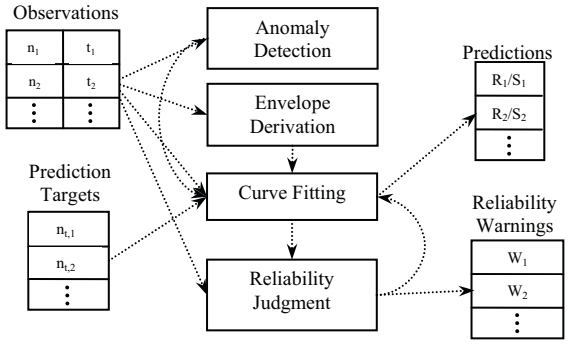


Figure 2. ADEPT components. Arrows show information flow.

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.

We will first 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.

## V. Obtaining the Predictive Model with ADEPT

### A. *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 $\delta$ 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 pairwise 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 percentage of deviation $\delta$ 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, $\delta$ 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: $\delta \geq \max_{observations}\left(\left|T_{actual}-T_{predicted}\right| / T_{actual}\right)$. If this test fails, our initial guess for $\delta$ was incorrect, and $\delta$ 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 $\sigma$ values, respectively: $E=\left\{c_i=\left(\left[A_{i,\min},A_{i,\max}\right], \left[\sigma_{i,\min},\sigma_{i,\max}\right]\right) \middle| i=1,...,K\right\}$. 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 3). The envelope consists of all model 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 to those that can describe the observations if assuming that no observation deviates from the model by more than $\delta$.

In model-parameter range calculation, we have derived closed-form formulas and their extensions using $\delta$ 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 [22]. 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 $\delta$ parameter included:

$$\sigma = 2\left(n_j t_j - n_i t_i\right)/\left(n_j - n_i\right) \quad (1)$$

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

$$\sigma \in \left(2\left(n_j t_j - n_i t_i\right)/\left(n_j - n_i\right)\right)\left[1-\delta, 1+\delta\right] \quad (3)$$

$$A \in \left[n_i t_i(1-\delta) - \sigma\left(n_i-1\right)/2,\ n_i t_i(1+\delta) - \sigma\left(n_i-1\right)/2\right] \quad (4)$$

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 following actions are performed to establish the envelope:

- Use the set of observations, $I=\left\{\left(n_i,t_i\right) \middle| i=1,...,M\right\}$, to form all pairs of observations, $P=\left\{\left(\left(n_i,t_i\right),\left(t_j,t_j\right)\right) \middle| \forall i,j \in \{1,...,M\}, i<j\right\}$, where $M$ is the number of observations.
- For each observation pair $p \in P$, calculate all possible range pairs for the parameters, $C_p=\left\{\left(\left[A_{i,\min},A_{i,\max}\right],\left[\sigma_{i,\min},\sigma_{i,\max}\right]\right) \middle| i=1,...,k_p\right\}$. 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 $c_l=\left(\left[A_{l,\min},A_{l,\max}\right],\left[\sigma_{l,\min},\sigma_{l,\max}\right]\right)$ is redundant if there exists some $c_j=\left(\left[A_{j,\min},A_{j,\max}\right],\left[\sigma_{j,\min},\sigma_{j,\max}\right]\right)$ such that $\left[A_{l,\min},A_{l,\max}\right] \subseteq \left[A_{j,\min},A_{j,\max}\right]$ and $\left[\sigma_{l,\min},\sigma_{l,\max}\right] \subseteq \left[\sigma_{j,\min},\sigma_{j,\max}\right]$. 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

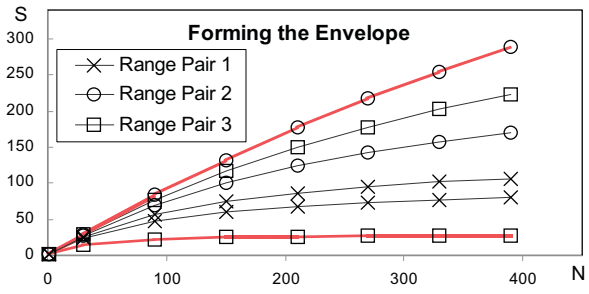


Figure 3. Forming the envelope. Range Pair 1 is redundant and discarded. Range Pairs 2 and 3 are combined to form the envelope, with absolute bounds shown via heavier lines.

TABLE 2. COMPARISON OF RUNTIME OF ADEPT WITH THREE CURVE FITTING METHODS: 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 | $\sigma$ 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 |

means discarding $c_l$ while keeping $c_j$ would not modify the set of model instances that are examined, as shown in Figure 3. The result of discarding those range pairs which are redundant is the final set of range pairs $E = \left\{ c_i \mid i = 1,...,k,\ k \le |C'| \right\}$.

### B. *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 for all predictions. Each model instance is specifically biased towards its corresponding prediction target. This is accomplished by giving more weight to the more relevant, i.e. closer, observations. The motivation is that, for extrapolative speedup prediction, the closest observation typically best shows the trend.

The input to curve fitting are the observation points, $I = \left\{ (n_i, t_i) \mid i = 1,...,M \right\}$, the envelope to which the search is limited, $E = \left\{ c_i \mid i = 1,...,k \right\}$, and the number of nodes on which a prediction is needed, $n_{\text{target}}$. 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, $(A_{final}, \sigma_{final}) = CurveFitting(I, R, n_{\text{target}})$.

Our optimality criterion for curve fitting is the Weighted Sum of Squared Relative Errors (WSSRE). The weight of a point is calculated as: $W_i = q * \max\left\{ |n_{target} - n_j| \mid j = 1,...,M \right\} - |n_{target} - n_i|$, where the factor $q$ determines how sensitive the weights will be to prediction distance, with smaller values being more sensitive.

The exhaustive search is performed in two steps 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. By fixing one point, we can calculate values for $\sigma$ from any value of $A$, and we perform exhaustive search only on those values of $A$ which fall within the envelope. This one-dimensional search finds a good model instance in linear time. To obtain the final model instance, we then perform two-dimensional exhaustive search, varying the $A$ and $\sigma$ values obtained in the one-dimensional search. The variation is done with fine-grain steps, modifying $A$ and $\sigma$ up to $\upsilon$ percent constrained by the envelope.

### C. *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 [23], a common optimization approach. We used its implementation levmar [24], with default settings, and arbitrary initial guesses for the parameters. The genetic algorithm implementation used is GALib [25]. Boundaries within which to search were set as $A$: 1 to 3,000, and $\sigma$: 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 4 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 $\sigma$ for up to 1000 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, with less than 100 ms and less than 500 ms runtimes, respectively.

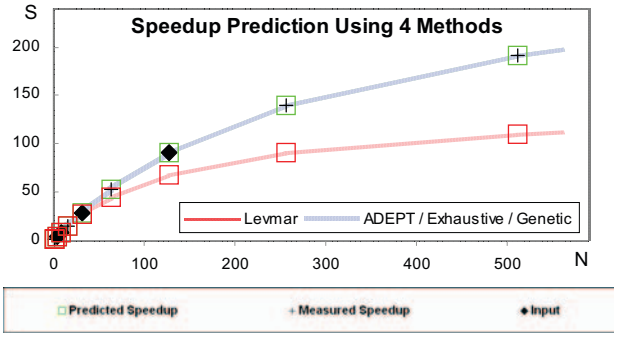


Figure 4. Comparing speedup prediction for ADEPT, genetic algorithm, exhaustive search, and Levenberg-Marquardt. The first three made perfect predictions (higher trend line), while the fourth was inaccurate (lower trend line).

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.

## VI. 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 [13]. We ran these benchmarks on clusters of SHARCNET [26], with three runs per benchmark and per number of nodes. Each MPI benchmark was run with one process per multi-core node. 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 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 used estimated *T(1)* to draw speedup curves. When *T(1)* was available, we omitted *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 $E = \left( \left| predicted - actual \right| / actual \right) * 100$ 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 $\sigma$ 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 parameter $\upsilon$ was set to 15 (which means 15% variation); higher values did not result in any accuracy improvements.

## VII. Experimental Results for Model Derivation

### A. Speedup Prediction

We first demonstrate the performance of ADEPT in speedup prediction for normal cases. The results are shown in Figure 5. 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 did not include *T(1)* in the input, and the second did. *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 was calculated using the predicted *T(1)* and does not show the actual speedup. Thus, the envelope may not contain all the observations in the speedup space (as e.g. for NAS_CG in Figure 5). 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 did 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 showed an error of 47% at 128 nodes when a maximum of 32 nodes was 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.

Additional experiments investigated placement and using more (up to 6) 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 in Section 9). More input points also did not generally result in any accuracy improvement.

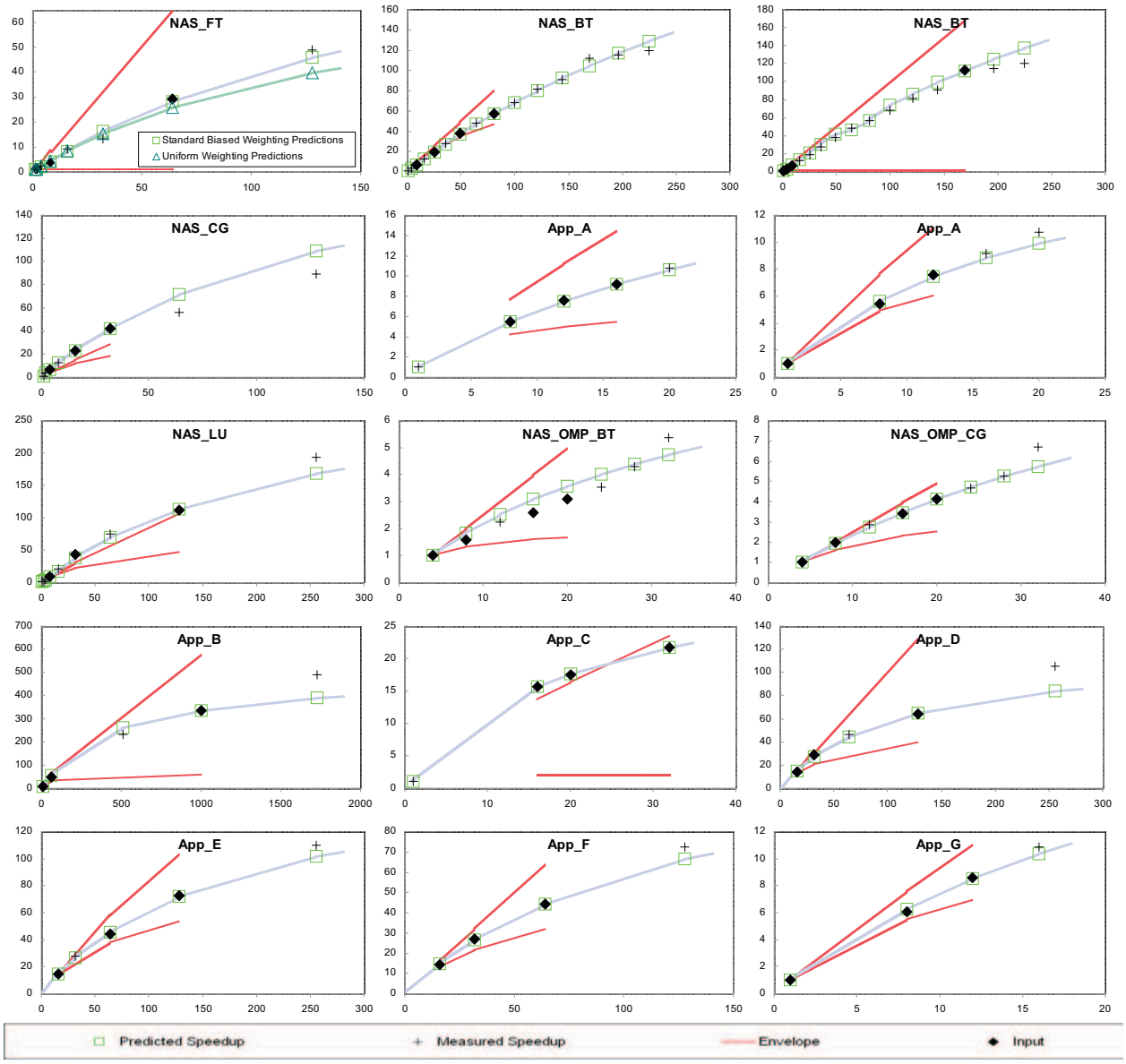


Figure 5. Speedup prediction results for the MPI implementation of NAS BT, CG, FT, and LU, the OMP implementation of NAS BT and CG, and anonymous real applications, both interpolation and extrapolation. Graphs show *S* over number of threads for OMP benchmarks, *S* over *N* otherwise.

## B. *Runtime prediction*

Next, we demonstrate the performance of ADEPT at runtime prediction. Results are shown in Figure 6. Due to space limitations, results for App_A, App_C, and App_G are not shown. However, as shown in Figure 5, 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 were so accurate that measurements and predictions would overlap on the runtime curve, and hence we omitted 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 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 80%, with the exceptions 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

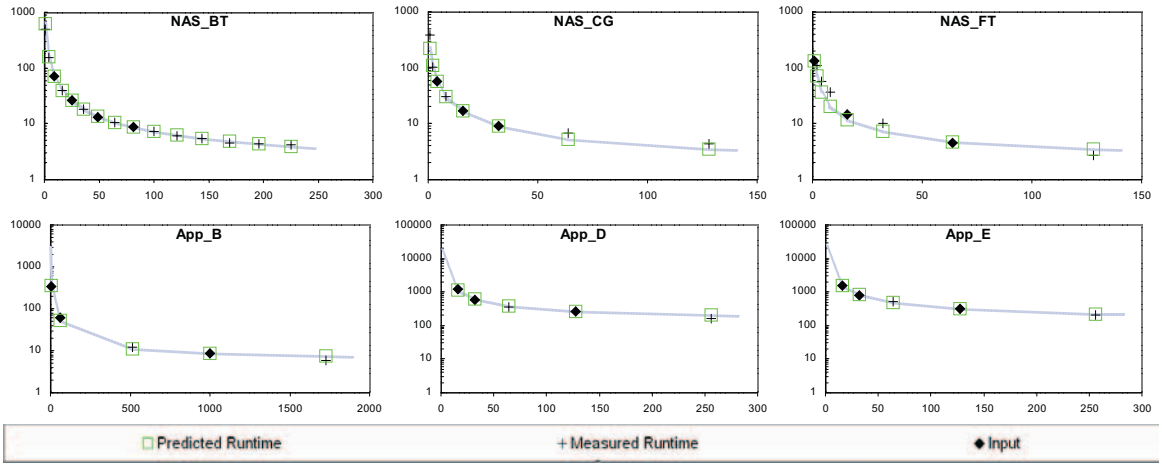


Figure 6. Runtime prediction results for NAS benchmarks and anonymous real applications, for both interpolation and extrapolation. Graphs show T over N.

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 did not show any significant improvement in accuracy of predictions for any benchmark or real application. The only trend proven, as a byproduct of these 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 (we only found two approaches which also use the NAS benchmarks with a similar range of node counts[1]), results in [18] 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 achieved more than 90%, and 82%. The accuracies for the very complex white-box approach presented in [11] were about 90% for SP, about 90% for BT, and 80% to 90% for LU. For certain cases, the results showed lower accuracy (e.g. 75% for LU). Our proposed method achieved 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. Our proposed method also outperforms the white-box approach in [20], which shows generally 80% accuracies for BT, CG, FT, and LU. 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.

[1] Most white-box approaches evaluate their work with special and differing real-life applications.

## VIII. Anomaly Detection

### A. 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, $\varepsilon$, which specifies the percentage of increase in *R* that is to 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 can not 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 7 (bottom row, center) the interpretation of a declining phase is chosen, and no predictions are made for this point.

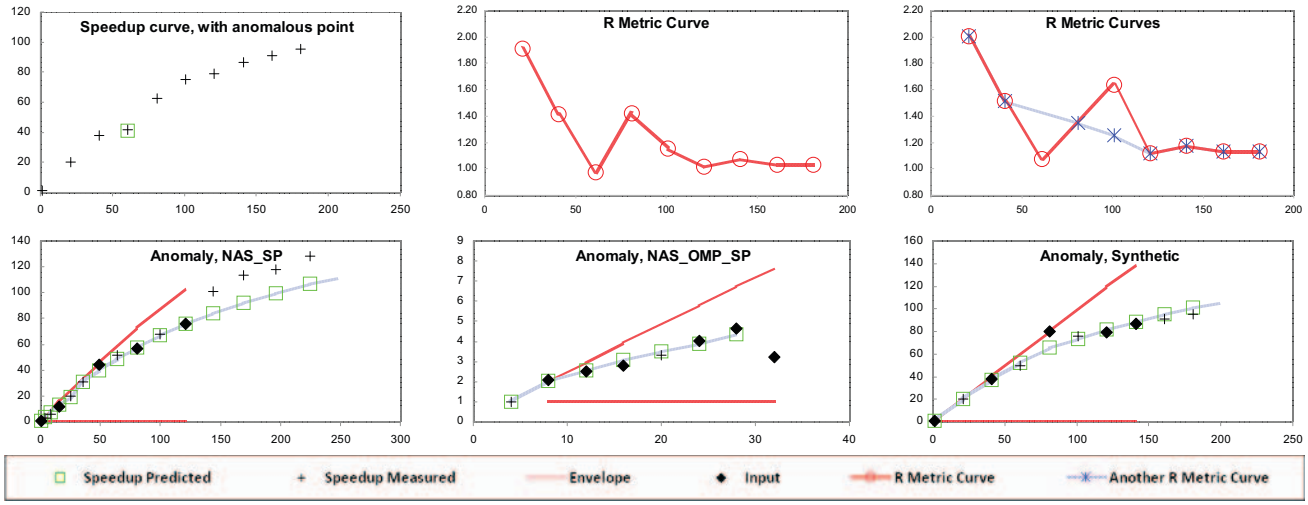


Figure 7. Detection and handling of individual anomalous points. Top row shows a synthetic speedup curve with an anomalous point (boxed), the associated *R* metric curve, and the *R* metric curves resulting from removal of anomaly candidates. The bottom row shows experiments integrating the anomaly detection step into ADEPT: speedup curve predicted by reducing the weight of anomalous point at 49 nodes (left); predicted speedup curve for NAS_OMP_SP by adjusting the weight of anomalous point at 16 nodes (middle); and speedup curve predicted in spite of the anomalous input point at 80 nodes (right). Top row left and bottom row plot *S* over *N*, or *S* over number of threads for OMP. Top row middle and right plot *R* metric over *N*.

### B. *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 7 (top row), 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 $D_l = \left(R_{j+1} - R_j\right)/\varepsilon$.

Individual anomalous points and their corresponding magnitude of deviation are reported to the curve fitting component described in Section 4, which reduces the impact of anomalous points on curve fitting via adjusting the weight of each anomalous point j as: $W_j^{'} = \max\left(0, W_j * \left(\theta - \min\left(\phi, D_j\right)\right)/\phi\right)$. $\varphi$ is set as the maximum deviation magnitude allowed for a point that impacts curve fitting. The deviation tolerance threshold $\theta$ describes how much anomalous points can affect the curve fitting, with higher values meaning more impact. $\theta$ can be set to any value between 1 and $\varphi$. In the experiments involving anomaly detection, we set $\varepsilon$ to 0.1, $\varphi$ to 10 and $\theta$ to 5. We found these values to be optimal in detection of anomalies.

During the testing of ADEPT for the NAS benchmarks, in several cases, anomalous points were identified and given reduced weight. See Figure 7 (bottom row) 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.

### C. *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 superlinear 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 8 (top row) 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

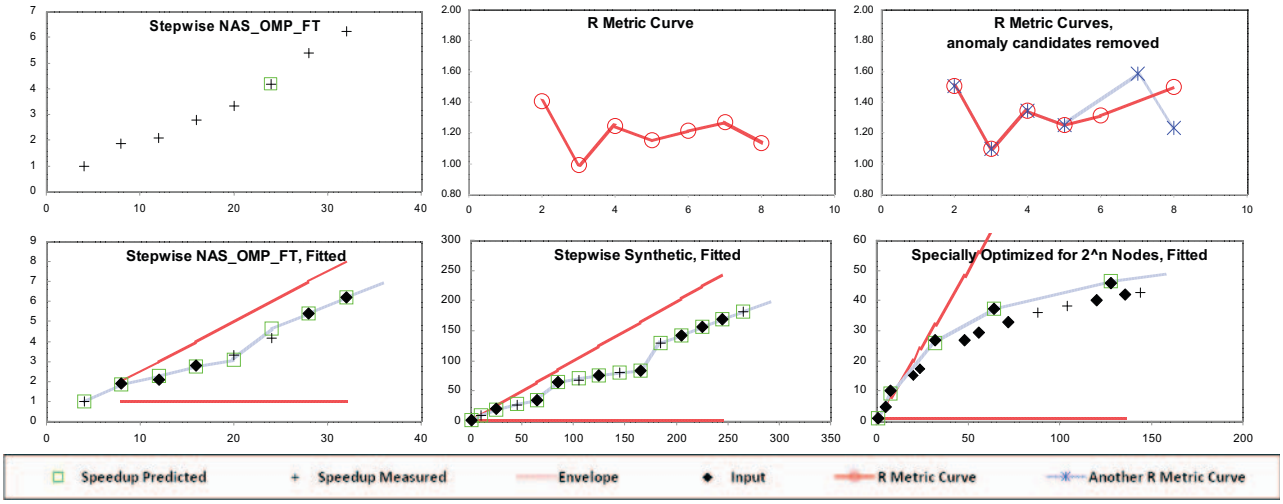


Figure 8. Detection and handling of specific scalability patterns. Top row: Speedup curve for runtimes used as input (left), with step highlighted. R metric curve for all input points (center). R metric curve with candidate points at 6 CPUs removed (stars) and at 7 CPUs removed (circles). Bottom row: Resulting prediction from example in top row (left), additional example of stepwise speedup with three phases (middle) and specially optimized application (right). Left column shows S over number of threads, top row middle and right plot R over N, bottom row middle and right show S over N.

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: $\forall(n_i,t_i)\in I: n_i < A$<br>High variance model has no linear section | Request additional observation on $\geq A$ 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) |

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 [2]. 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 8. For the first stepwise case (OMP_FT), ADEPT identified the change of program phase between 5 and 6 CPUs, and chose 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 Stepwise Synthetic.

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.

## IX. 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.

High fitting error is the simplest case. Input points are not identified as anomalous but experience large fitting errors (>10%). See LU in Figure 9 (left); the point at 32 nodes experienced 16% error in speedup (18% error in runtime). ADEPT 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 9 (middle) for an example where the input points all lie in the linear section. In this case, ADEPT suggested 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

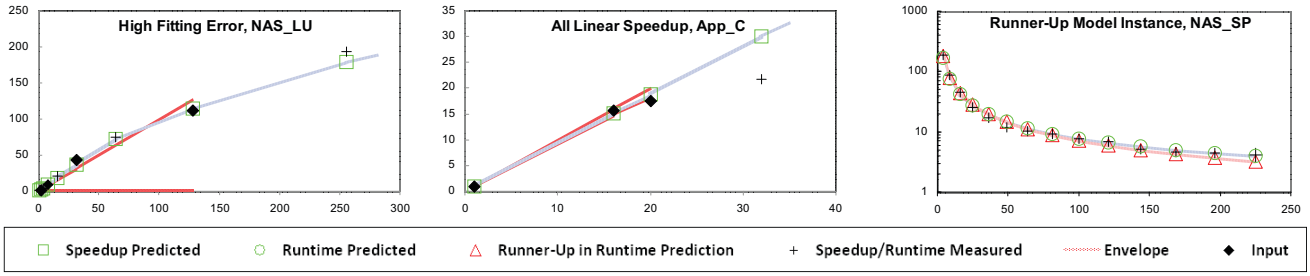


Figure 9. Reliability judgment due to a high fitting error, all inputs in the linear section of speedup curve, and the existence of runner up instances. Left and middle graphs show S over N. Right graph shows T over N in log scale.

greatly different *A*. An example is shown in Figure 9 (right), where the fitted model instance for prediction target at 49 nodes had a value of 700 for *A* parameter, and there was a runner-up instance with a value of 320 for *A*. This occured because, as shown in the graphic, the runtime values for the two model instances converge near the input points. The difference between runtime values of the two instances was 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 that differ by 20%, resolved the problem by identifying the runner-up instance as the best fit.

## X. Summary and Conclusion

We have presented a black-box approach, based on the Downey model, for prediction of 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 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. This makes ADEPT suitable for adaptive resource allocation in production environments based on efficiency curves and for extrapolative scalability prediction as feedback to users.

## Acknowledgment

This research was funded by a SHARCNET Graduate Fellowship. 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, and Wai Ling Yee for hints in regards to the closed-form solution to the Downey model.

## References

[1] 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.

[2] 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.

[3] 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.

[4] 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, June 2006, Springer.

[5] 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, May 2009, Rome, Italy, to appear in Springer.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

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

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

[13] 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.

[14] R.A. Gibbons. Historical Application Profiler for Use by Parallel Schedulers. Proc. IPPS Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), LNCS 1291, Springer, 1997.

[15] W. Smith, I. Foster, and V. Taylor. Predicting Application Run Times with Historical Information. J. Parallel and Distributed Computing 64, 2004, pp. 1007-1016.

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

[17] A. Duran, J. Corbalan, and E. Ayguade. An Adaptive Cut-off for Task Parallelism. Proc. ACM/IEEE Supercomputing (SC), 2008.

[18] 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.

[19] B.J. Barnes, B. Rountree, D.K. Lowenthal, J. Reeves, B. de Supinski, and M. Schulz. A Regression-based Approach to Scalability Prediction. Proc. Internat. Conf. on Supercomputing, 2008.

[20] 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.

[21] 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.

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

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

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

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

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