A Modeling Approach for Estimating Execution Time of Long-Running Scientific Applications

Seyed Masoud Sadjadi[1], Shu Shimizu[2], Javier Figueroa[1,3], Raju Rangaswami[1], Javier Delgado[1], Hector Duran[4], Xabriel J. Collazo-Mojica [5]

1: Florida International University (FIU), Miami, Florida, USA; 2: IBM Tokyo Research Laboratory, Tokyo, Japan; 3: University of Miami, Coral Gables, Florida, USA; 4: University of Guadalajara, CUCEA, Mexico; 5: University of Puerto Rico, Mayaguez Campus, Puerto Rico;

Abstract

In a Grid computing environment, resources are shared among a large number of applications. Brokers and schedulers find matching resources and schedule the execution of the applications by monitoring dynamic resource availability and employing policies such as first-come-first-served and back-filling. To support applications with timeliness requirements in such an environment, brokering and scheduling algorithms must address an additional problem - they must be able to estimate the execution time of the application on the currently available resources. In this paper, we present a modeling approach to estimating the execution time of long-running scientific applications. The modeling approach we propose is generic; models can be constructed by merely observing the application execution "externally" without using intrusive techniques such as code inspection or instrumentation. The model is cross-platform; it enables prediction without the need for the application to be profiled first on the target hardware. To show the feasibility and effectiveness of this approach, we developed a resource usage model that estimates the execution time of a weather forecasting application in a multi-cluster Grid computing environment. We validated the model through extensive benchmarking and profiling experiments and observed prediction errors that were within 10% of the measured values. Based on our initial experience, we believe that our approach can be used to model the execution time of other time-sensitive scientific applications; thereby, enabling the development of more intelligent brokering and scheduling algorithms.

1. Introduction

A Grid computing environment provides a shared resource infrastructure for a large number of applications. Entities such as brokers and schedulers decide how resources get partitioned among the set of applications. Typical policies employed by these entities include back-filling, first-come-first-served, etc. and are based on dynamically monitoring application resource usage behavior. While such policies work reasonably well to balance application resource requirements across physical resources and ensure high resource utilization, they fall short when the scheduling task involves meeting task-specific execution deadlines. To address timeliness requirements, brokers and schedulers must be able to estimate the execution time of the application on the currently available resources in order to ensure that scheduling decisions do not lead to deadline violations.

Extensive research exists in the area of resource usage and execution time prediction. A large section of this work focuses on platform-specific approaches [1,2,3,4,5], which does not support resource usage prediction on previously "unseen" target execution environments and very few of these approaches address performance prediction across different hardware configurations. This is important since the set of available resources on the Grid could have an arbitrary configuration. Some approaches that do provide support for cross-platform prediction of resource usage such as [6] and [7] are either application specific or restricted to predicting for single-node application executions alone. Other approaches are intrusive in that they call for application source or binary instrumentation.

Our approach differs from the other works in several respects. First, our approach is application agnostic and does not require application source or binary code inspection or instrumentation. Second, unlike some previous approaches, our approach does not require a sample execution on the target platform before prediction. Third, our approach is able to model execution scale, thereby also addressing distributed applications; especially, it allows prediction in a multi-cluster Grid computing environment.

A key contribution of our approach is the generality of our application resource usage model, which is a direct result of constructing the model by merely observing the application execution "externally" (i.e., observing its resource usage rather than inspecting or instrumenting its code). This allows a system design and implementation that is completely oblivious of the semantics of the target application. The advantage of such an application-agnostic approach is more appreciated when application semantics are complex, the source-code is not available, or there is no documentation. The WRF (Weather Research and Forecasting) application is an example of a

978-1-4244-1694-3/08/$25.00 ©2008 IEEE

time-sensitive, resource-intensive, distributed application with a complex codebase and little to no documentation.[1] We use it to demonstrate the effectiveness of our approach through the rest of the paper. Once a cross-platform application model is built, predicting the execution time on a different hardware configuration simply involves populating the correct values for individual resources of the new platform.

The proposed model estimates the application execution time based on application resource usage behavior. We note that the application execution time may be dependent on the specific configuration of several resource types. We started with a simple first step assumption that the influence of a single resource on the execution time of the application is independent from the influences of the other resources. We concede that this assumption may not be valid and some resource correlations may influence application execution time and intend to relax this assumption in future extensions of this work. We shall demonstrate, even with the simplistic assumption, that the model we developed is fairly extensive, capable of incorporating multiple resources and multi-node application executions.

The effectiveness of our approach was tested by developing a model for predicting the execution time of WRF in a multi-cluster environment. Several experiments were done to validate the model. An error rate of less than 10% was observed, which leads us to believe that the model is a viable option for developing timeliness enhancements to brokering and scheduling applications that take into account the dynamic resource availability of target execution environments. It is worth pointing out that we have applied the model proposed in this paper to other applications with substantially different resource consumption characteristics and have found it to be effective [8].

The rest of this paper is organized as follows. In Section 2, we describe the general approach taken by our model, including the parameters we are modeling. In Section 3, we introduce the mathematical model being evaluated. Section 4 describes the software artifacts that we developed for implementing our modeling and prediction mechanisms. This section also describes how this software was used to carry out our experiments to evaluate the model. Section 5 shows the results obtained from the experiments. Section 6 describes how the model was validated, based on the accuracy of the results obtained. Section 7 provides a more in-depth look at related research (compared to that which has already been covered above). Finally, in Section 8, we summarize the paper and provide some future research directions.

[1] Several of such applications exist in the domain of scientific computing and elsewhere. Ironically, these applications are also often mission-critical.

2. Approach to Modeling Resource Usage

Modeling the resource usage of a distributed application must take into account several aspects of the computational environment. In this section we overview our approach to modeling the static resource properties of the target execution platform and execution-specific factors affecting resource usage such as the degree of parallelism. Model construction is formally addressed in Section 3.

To construct our model, we build upon our initial work on modeling single-node application execution [8]. Following the philosophy of the original approach, we construct the model to be application-agnostic (as opposed to application-specific). While profiling using source-code instrumentation (an application-specific approach) can provide valuable insight into application behavior that is typically unavailable with external observation, our application-agnostic approach provides generality in the modeling, profiling, and prediction mechanisms. Consequently, this enables a system design and implementation that is completely oblivious of the semantics of the target application. The latter is especially important when application semantics are either complex, or are unknown, or if the source-code is unavailable.

2.1 Modeling the Resources

Resource properties of the execution platform fall into the three basic categories of *computation*, *communication*, and *storage*. The key computational resources that affect execution time include the CPU clock speed, L2 cache size, and the front-side-bus (FSB) bandwidth. Communication parameters include the maximum bandwidth and latency of the network interconnection, while storage parameters include the main memory size and memory access bandwidth, as well as sequential and random disk I/O bandwidths.

While the above set of parameters may seem excessive, we point out that the significant parameters of the model that affect the performance of a specific application is typically a subset. Our modeling technique automatically identifies these parameters; the remaining parameters are typically eliminated from the model with sufficient evidence. In the rest of this paper, however, we restrict our WRF model to a subset of computation resources by modeling the CPU clock speed and the number of nodes, and ignore the influence of other resource properties for the sake of simplicity of exposition.

2.2 Modeling Execution Parallelism

When modeling parallel and distributed applications (as is typical for scientific applications), the two critical parameters to address are *platform heterogeneity* and *execution scale*.

In our current model, we assume that all the nodes in the target execution environment are identical, i.e. that they have identical individual resource characteristics of computation, communication, and storage. While we realize that this may not apply to all distributed environments, it is effective in addressing typical cluster environments. More importantly, it allows us to construct a practically usable model of the system. Relaxing this assumption to accommodate heterogeneous clusters is an important direction for future work.

We address execution scale by including in our model a parameter associated with the number of processors utilized during execution. For simplicity, our model currently makes no distinction between shared and distributed memory processors (e.g., SMP and Cluster).

2.3 Modeling Input Parameters

Accurate input parameter modeling requires knowledge of application semantics. In the spirit of our application-agnostic approach, we simplify the modeling of input parameters by reducing input parameter modeling to a *load specification* task for the developer. The load is interpreted as linear, with higher values representing a greater input load on the application.

In case load specification is infeasible (due to the complexity of the application input data semantics), we consider the execution of an application with a different input data set as a "new application", which must then undergo independent profiling and modeling.

3. Application Resource Usage Model

In this Section, we present a model of application execution correlating with application resource usage characteristics. In constructing our model, we note that applications may utilize different types of resources (as elaborated in Section 2) and also demonstrate varied patterns of resource usage during their execution. Consequently, the execution time of each application is typically dependent on different sets of resources. To take this into account, we create application profiles to capture and predict execution time for a specific application. As a first step assumption, we suppose that the influence of the resource properties of the target execution environment to the application execution time takes a product form, in which each term represents the influence of one or more resource properties and is independent of other terms. It is therefore represented, as follows:

$$T_{exec} = \prod_{i=0}^{m-1} C_i \,, \tag{1}$$

where T_{exec} is the execution time, C_i is the *i*-th contribution by one or more resources, and *m* is the number of contribution terms. The term C_i may contain resource parameters such as *CPU* clock frequency, L2 cache size, FSB bandwidth and disk I/O bandwidth, as mentioned in Section 2.

In this paper, we consider and focus on two types of independent contributions: the parallelism of task execution and CPU's performance factor, which is currently just it's clock frequency.[2] In considering parallel task execution on more than one node, we further assume that all nodes are homogeneous in terms of their resource properties.[3] Further, the model more naturally addresses applications whose resource consumptions are more or less consistent across time. If this is not the case, the model would still identify all the dominant resource usage factors, but would however be unable to capture the dynamics of such usage over the duration of the application execution. We use the following simple form of the parallelism term, which reasonably assumes that the execution time is in inverse proportion to the degree of parallelism, or

$$C_0 = \alpha_0 + \alpha_1 / P_{para} \,, \tag{2}$$

where P_{para} is the degree of parallelism such as the number of processors available to the application, α_0 and α_1 indicate the application's characteristics and will differ for each application. The first term α_0 indicates the constant contribution such as execution overhead.

To include the CPU performance contribution, we use the following simple form:

$$C_1 = \beta_0 + \beta_1 / P_{clock} \,, \tag{3}$$

where P_{clock} is the CPU clock frequency, β_0 and β_1 indicate the application's characteristics related to the CPU performance as well as Equation (2).

After expanding the product form of Equation (1), we have a simple summation form, as follows:

$$\begin{aligned} T_{exec} \quad &= \alpha_0\beta_0 + \alpha_1\beta_0 / P_{para} + \alpha_0\beta_1 / P_{clock} \\ &\quad + \alpha_1\beta_1 / \left(P_{para} P_{clock}\right) \end{aligned} \tag{4}$$

which is a linear form of *explanatory variables* (basic terms for regression analysis) multiplied by application

[2] Please note that while a more exhaustive model would consider additional resource properties (e.g., memory size) or more complex dependencies (e.g., quadratic instead of linear), our primary goal in this paper is to introduce our general technique for resource usage modeling and prediction. Therefore, we chose a minimal model to simplify presentation. We minimize the effect of these parameters by making them invariable in all experiments.

[3] While some cluster environments would consist of a set of machines of almost similar configuration, there may be cases where this assumption may not hold. Relaxing this assumption, however, substantially complicates the modeling mechanism.

profile parameters that define application characteristics. We can denote it by its following general representation:

$$T_{exec} = \theta_0 + \sum_{i=1}^{n} \theta_i y_i \quad (5)$$

θ_i is the *i*-th application *profile parameter* which are values reflecting constant contributions related to a particular application and type of resource and y_i is the *i*-th *explanatory variable* which is a function of static resource properties (e.g. clock speed) and the resource competition status.

Equation (5) thus reduces the prediction task to a problem of estimating profile parameters. We apply regression analysis to solve the problem. While the general technique we use, including the error analysis, is detailed in [8], we summarize key steps of the estimation process here. The observation at time *t=k* includes the monitored execution time $x^{[k]}$ on the static resource set:

$$\mathbf{y}^{[k]} = \begin{bmatrix} 1 & y_1^{[k]} & \cdots & y_n^{[k]} \end{bmatrix} \quad (6)$$

After obtaining *N* sets of observations, Equation (5) is now represented with an error term, as follows:

$$\mathbf{x} = \mathbf{H}\boldsymbol{\theta} + \boldsymbol{\varepsilon} \quad (7)$$

where

$$\mathbf{x} = \begin{bmatrix} x^{[0]} & \cdots & x^{[N-1]} \end{bmatrix}^T$$

$$\mathbf{H} = \begin{bmatrix} \mathbf{y}^{[0]} & \cdots & \mathbf{y}^{[N-1]} \end{bmatrix}^T$$

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 & \cdots & \theta_n \end{bmatrix}^T$$

and ε indicates errors with zero mean. The error term may include observation errors and model inaccuracy. We then apply regression analysis to minimize the mean square errors of ε to estimate the application profile parameters, as follows:

$$\hat{\boldsymbol{\theta}} = \left(\mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{x} \quad (8)$$

Note that the matrixes *H* and **x** grow in size for more observations, however, the matrixes $H^T H$ and $H^T x$ are of the fixed size of $(n+1)\times(n+1)$ and $(n+1)\times 1$, respectively. As a result, we do not need additional storage to maintain the observations in general. It should be also noted that the matrix $H^T H$ must have a valid inverse matrix to obtain a solution to the above equation. Thus, at least *n+1* sets of independent observation data are required for realizing a model of the application. For reliable estimation, more than *n+1* observation must be maintained so as to decrease errors of observation.

4. Monitoring and Prediction

For profiling applications, we have developed two software programs: (1) a monitoring program (called *amon*) that runs on each compute node and reports the execution time observed for any application with appropriate values of explanatory variables every time it detects completion of the application's execution; and (2) a prediction program (called *aprof*) that runs on a server node and receives the reports from the monitoring programs distributed over the compute nodes. It then maintains the fixed-size matrixes $H^T H$ and $H^T x$ for each application in an incremental manner. For example, each element a_{ij} of $H^T H$ is updated, as follows:

$$a_{ij}^{[k]} := a_{ij}^{[k-1]} + y_i^{[k]} y_j^{[k]} \quad (9)$$

where $a_{ij}^{[k]}$ is the element value ($0 \le i, j \le n$) at time *t=k*, $a_{ij}^{[0]} = 0$ at *t=0,* and $y_0^{[k]} = 1$. Elements of $H^T x$ are also calculated in an incremental manner without increasing the data size in the profiling program.

After the profiling program receives sufficient number of independent observation results, that is, at least *n+1* sets of data, it can estimate and calculate the profile parameters by using Equation (8).

Once the profile parameters for a certain application are estimated, then we can apply them to predict an execution time for a certain resource, or

$$\hat{T}_{exec} = \begin{bmatrix} 1 & y_1 & \cdots & y_n \end{bmatrix} \begin{bmatrix} \hat{\theta}_0 & \hat{\theta}_1 & \cdots & \hat{\theta}_n \end{bmatrix}^T \quad (10)$$

where y_i is the value of the *i*-th explanatory variable in the given resource set and $\hat{\theta}_i$ is the *i*-th estimated profile parameter for the application. Thus, the prediction program not only estimates application profile parameters but also applies them to predict an execution time based on an explanatory variable set ($[y_1, \ldots, y_n]$).

The *amon* and *aprof* were designed to run in a networked environment using a client/server architecture. We run one instance of *aprof* on the head node of the cluster, which is being profiled, to act as the server and run one instance of *amon* on each compute node of the cluster to act as clients. Due to our need to execute tests for a number of different nodes and for a range of *CPU* speeds (as will be explained in Section 5), it was necessary to automate the experiment executions. Various perl, bourne shell, and python based scripts were created to execute these simulations automatically. The basic functionality of these scripts is to gather the results in the form of *amon* output and turn them into *aprof* compatible prediction input to later compute the predictions that will prove the accuracy of our model.

5. Experiments and Results

Theoretically, by adding more processors to a simulation one could assume that a scientific application could run faster. This assumption is challenged by the existence of a predicted point on the performance curve where the running time of the application will worsen as more processors are added to the system (i.e. the saturation point). Several computational variables (e.g. clock speed of the processor, processor load, number of compute nodes, memory, network latency, network bandwidth, communication overhead) determine the execution time. In this section, we present a number of experiments that help us analyze the behavior of WRF by modifying two of the resource parameters, namely, processor load and number of compute nodes.

To limit the number of concurrent changing variables, we assume that the execution time of a forecast simulation is based only on changes in clock speed and number of compute nodes. For this, we have kept the effect of other parameters such as memory minimized by making sure that the value of the parameter either does not change or is above an upper bound on the observed and target platforms. The experiments were conducted using two compute clusters located at Florida International University. The first cluster is called *GCB*. This cluster is based on the NPACI Rocks Linux distribution for compute clusters, version 4.0. The cluster contains 8 nodes where each node contains two 32-bit x86 Intel based CPUs, 1GB of main memory and uses a gigabit network connection. The second cluster is called *Mind*. *Mind* is also operating on *Rocks* version 4.0. The cluster consists of 15 nodes, each containing dual Xeon 3.6GHz processors and 2GB of main memory. They are also connected through a gigabit network connection.

The WRF application running in a cluster environment is capable of distributing the domain data points of a forecast simulation into optimal size; this is called optimal domain decomposition and is executed by WRF communication RSL [9] layer. The code uses MPI [10] communication subroutines for inter node communication, and OpenMP [11] for intra node, inter processor communication. These subroutines enable the program to distribute the data and computational load among the nodes of the cluster and the processors inside each node, respectively. For the purpose of these experiments we have used a small domain configuration to minimize the size of our cases to match the size of our clusters' computational power, while the effect of other resources (e.g., memory size) will be minimal. As suggested by meteorologists, we used a 75 by 75 domain decomposition with 4km resolution, which contains 5625 grid points.

The approach to the experiments was to benchmark the clusters by running experiments on different numbers of available compute nodes as well as different effective processing power. We employed three tools to assist us in our experiments. To limit the effective clock speed of the compute nodes, the open source *CPUlimit* [12] tool was used. The other two tools, mentioned in Section 4, are, *amon* and *aprof*. *Amon* was used to output run-related resource-consumption statistics of WRF simulation processes. *Aprof* was used for both receiving resource-usage characteristics of the WRF simulations and predicting the execution time for each of the simulations, based on *amon* output. The percentage values used for the CPU bindings were 100 (full utilization), 80, 60, 40, 30, 20, and 10 percent. All possible number of compute node utilized (i.e. from one to seven in *GCB* and from one to 15 in *Mind*). For example, if the experiment being conducted was 80-percent CPU bound then *CPUlimit* was run in every node that was part of the experiment using 80 as the "limit" parameter and *wrf_arw_DM.exe* as targeted process. The following figures show the results.

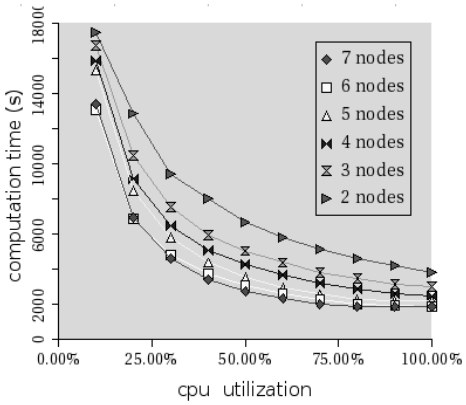

Fig. 1. The execution times of WRF on *GCB*.

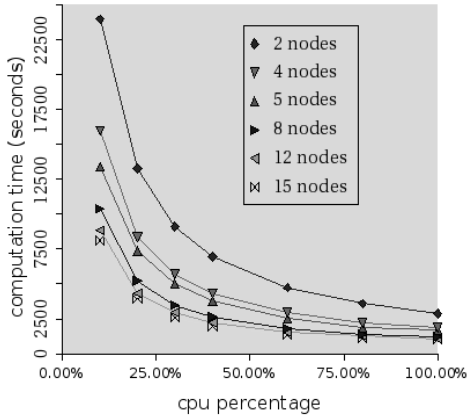

Fig. 2. The execution times of WRF on *Mind.*

Fig. 1 and Fig. 2 illustrate the curve obtained from limiting the processor power for the same WRF forecast simulation for different combinations of compute nodes and clock speeds on *GCB* and *Mind*, respectively. The performance decreases with less CPU power, but not linearly. Fig. 3 illustrates the relationship between the

inverse of the available *CPU* clock (i.e. *CPU* utilization) and the execution time, ranging from 2 to 15 nodes for *Mind*'s benchmark data. In the case of the inverse of the CPU speed, the experiments confirm our assumptions about the linear behavior that our resource usage model captures in Equation (3). We obtained a similar linear behavior in the case of the inverse of the number of nodes (not shown for the sake of brevity) and the same characteristics were observed in *GCB*'s output.

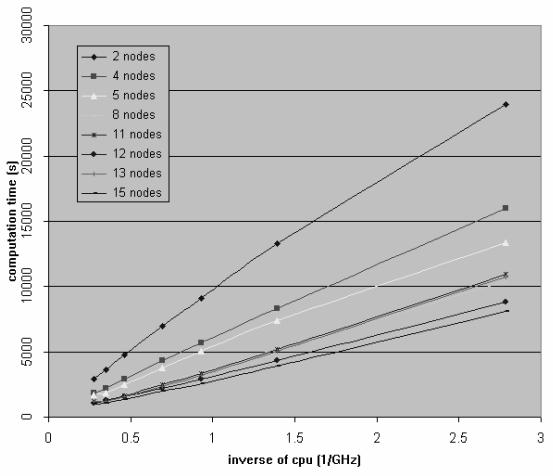

Fig. 3. The execution times plotted based on the inverse clock speed on *Mind* demonstrates a linear performance curve.

6. Model Validation

To validate our experiments and results, we used *aprof*. The mathematical model implemented in *aprof* is based on the fact that execution time decreases linearly along with the inverse of total computational power (*tcp*). Where *tcp* is equal to the product of clock speed and number-of-nodes. The *amon* results obtained from the previously exposed benchmarks were used as input data for the *aprof* program to predict different execution times of a WRF execution and calculate the accuracy of the prediction.

The model was first validated for within-a-cluster predictions on each of the two clusters. For each series of benchmarks (i.e. CPU-utilization and number-of-nodes combination) the actual execution time was compared to the predicted execution time. In *GCB*, the observed within-a-cluster fractional error rate was 5.34%. The median error rate was 5.86%. For *Mind*, the fractional error rate was 5.66% and the median 3.80%. Note that the prediction model uses the set of benchmark data (i.e. input data) as a database for improvement of accuracy in its predictions and as seen in the next prediction results, the greater the size of this data set the smaller the average fractional error.

For the across-clusters predictions, we used the same paradigm, this time using the statistics from *GCB* simulations (as inputs to *aprof*) to predict the execution time in *Mind*, and vice versa. When using *GCB*'s statistics to predict *Mind*'s execution time, we observed an average fractional error of 9.97% and a median of 5.86%. When using *Mind*'s input to predict *GCB*'s execution time, an average of 5.83% fractional error and a median of 4.13% were observed.

Our results demonstrate the validity of our model, despite its simplicity. Further research and validation will optimize our model, to account for differences in architecture (32- vs. 64-bit), and many other variables that might affect the accuracy of the model predictions. We believe that by having conducted these interpolation and extrapolation prediction scenarios we are able to show the potential behavior of WRF based on CPU load and number of compute nodes for future Grid enablement of this application.

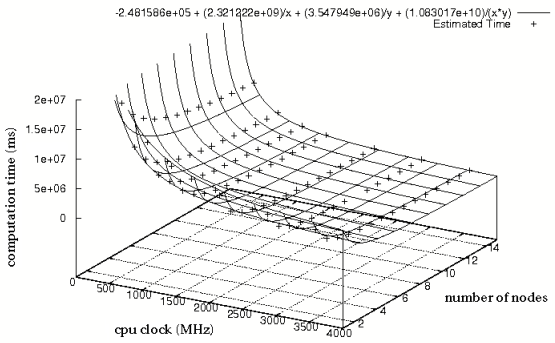

Fig. 4. The actual and estimated execution times for *Mind*'s predictions.

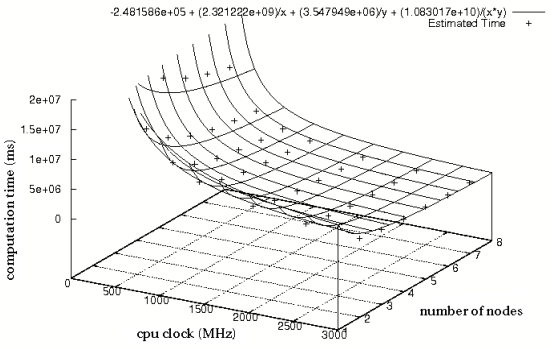

Fig. 5. Across-cluster predictions of *GCB*'s estimated time versus actual time, using *Mind*'s *aprof* input.

Fig. 4 portrays the actual values versus estimated values of the *Mind* benchmark series. It is possible to see that most of the high-error values occur for low number of nodes, which is not considered important in a cluster environment. Fig. 5 shows similar observations when comparing actual times versus *aprof*-estimated execution

times when using *GCB*'s input to predict *Mind*'s execution times.

Since our testbeds were using different architectures, we based our prediction tests and findings on number-of-nodes, rather than number-of-processors. *Mind* is using 64-bit Xeon processors with hyperthreading support, while *GCB* is using 32-bit Pentium 4 processors. For the purpose of the experiments, the logical (hyperthreading) processor was disabled on *Mind*. The fact that our results were as good as they are despite using different architectures reinforces the strength of our model.

7. Related Work

Extensive research has been carried out in the area of resource usage prediction. While a large section of this work focuses on platform-specific approaches [1,2,3,4,5], our work enables resource usage prediction on previously "unseen" target environments.

In the domain of Web services, the work defined in [13] proposes a method for online profiling of component-based services for predicting the response time. This work models each service individually in terms of its CPU utilization, CPU utilization for an RMI (Remote Method Invocation) operation, and network delay. Adapting our approach to multi-component and multi-platform web services can use the component-based profiling techniques proposed in the above work. However, a key general difference is that this approach is application-domain specific whereas the emphasis of our approach is to be application agnostic.

Predictions of finer granularity jobs, which may be useful for scheduling interactive applications, include a method to predict the running times of tasks [14]. The prediction method is based on the AR(16) model for CPU load estimations [15]. The work in [16] proposes an improvement on the AR(16) linear time-series model. Wolski *et al.* have focused on making short and medium term CPU load predictions [17]. Gibbons has proposed prediction methods targeting more general applications in which execution time predictions are obtained from the run times of similar applications [18]. He used templates to group similar applications. In later approaches [19, 20, 21], data mining techniques were used to search for good templates for a specific application. The PACE system [22] includes a method for predicting execution time and network usage, among others. Their method is based on both source code analysis and benchmarking analysis, quite opposed to the application-semantics agnostic philosophy of our work. In addition, one of the main differences of our modeling approach from most of the above is that our modeling framework allows for easy correlation of different resource configurations in conjunction with accounting for execution scale.

While there is abundant research on resource usage prediction in general, very few of these address performance prediction across different hardware configurations. Dimemas [23] is a performance prediction simulator which targets MPI applications. Performance prediction can be based on previous runs on different platform configurations. Different from this work our approach focuses on online prediction. Yang *et al.* propose cross-platform prediction by combining the application's performance in a reference system and the relative performance between the two systems derived from a partial execution on the target platform [6]. In their technique, the source code of an application is analyzed to identify the major time step loops and the source code is then modified to include the API for the partial execution measurements. Our work differs from this work in several respects. First, our approach, being application agnostic, is free from source-code instrumentation. Second, our approach does not require a sample execution on the target before prediction. Third, our approach is able to model execution scale, thereby also addressing distributed applications.

Marin and Mellor-Crummey [7] present a different approach for cross-platform prediction of application execution time. Their approach consists of statically analyzing the application binary code to identify the control flow graph for each routine as well as the loops contained. A dynamic analysis then obtains the frequency a routine is entered. Binary rewriting is used to augment an application to monitor and log information. Platform native instructions (e.g. SPARC instructions) are then translated into a set of generic RISC instructions. While this approach is more powerful than our proposed approach since it can address multiple architectures, this technique is restricted to predicting for single-node application executions alone. Our work allows prediction in a multi-cluster Grid computing environment.

8. Conclusion and Future Work

We have proposed a new approach for modeling the resource usage and execution time of a distributed application. The mathematical model we have proposed is a cross-platform model and can be constructed using observations external to the target application, requiring no inspection or modification of the application source or binary code. Experimental results using WRF executions on two clusters with different hardware configurations have demonstrated the efficacy of our approach for predicting the execution time of a long-running scientific application. Our cross-platform validation tests have demonstrated good accuracy (prediction errors within 10%), even with substantially different systems, using only two parameters of the execution environment, the number of nodes and the CPU clock speed.

Future work will include applying our model to other distributed applications as well as extending our model to include more resource parameters/contributors, such as cache/memory size, network bandwidth, and storage bandwidth. While we minimized the effect of these parameters by making them invariable in our current system, it is indeed important to validate our general approach to modeling resource usage on these additional non-trivial system resource properties. To target general Grid computing environments, we will also work on extending our parallelism model to address execution environments with heterogeneous resources. With regards to WRF in particular, we have come one step closer to devising a complete solution to our goal of higher-resolution weather prediction.

9. References

[1] S. Chen, I. Gorton, A. Liu, and Y. Liu. Performance prediction of COTS component-based enterprise applications. In Proceedings of CBSE5, 2002.

[2] M. V. Devarakonda and R. K. Iyer. Predictability of Process Resource Usage: A Measurement-Based Study on UNIX. In IEEE TSE, 15(12), December 1989.

[3] R. P. Doyle, J. S. Chase, O. M. Asad, W. Jin, and A. Vahdat. Model-based resource provisioning in a Web service utility. In USENIX Symposium on Internet Technologies and Systems, 2003.

[4] H. Kalva, R. Shankar, T. Patel, and C. Cruz. Resource estimation methodology for multimedia applications. In Proceedings of SPIE, 2007.

[5] D. M. Swany and R. Wolski. Multivariate Resource Performance Forecasting In the Network Weather Service. Proceedings of Supercomputing, November 2002.

[6] L. T. Yang, X. Ma, and F. Mueller. Cross-Platform Performance Prediction of Parallel Applications Using Partial Execution. Proceedings of Supercomputing, 2005.

[7] G. Marin and J. Mellor-Crummey. Cross-architecture performance predictions for scientific applications using parameterized models. SIGMETRICS Perform. Eval. Rev., 32(1):2–13, 2004.

[8] S. Shimizu, R. Rangaswami, and H. A. Duran-Limon. "Platform-independent Modeling and Prediction of Application Resource Usage Characteristics", Florida Int'l University Tech. Report FIU-SCIS-TR-2007-07-05, 2007.

[9] J.G. Michalakes. "RSL: A Parallel Runtime System Library for Regional Atmospheric Models with Nesting," IMA workshop on structured adaptive mesh refinement grid methods, Minneapolis, MN, 12-13 Mar 1997.

[10] Gropp, W., Lusk, E., Doss, N., Skjellum, A., "A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard", Parallel Computing, North-Holland, vol. 22, pp. 789-828, 1996.

[11] Dagum, L., Menon, R. "OpenMP: an industry standard API for shared-memory programming," Computational Science and Engineering, IEEE. Pages: 46-55, Vol: 5, Issue: 1, Publication Date: Jan-Mar 1998.

[12] Cpulimit - CPU Usage Limiter for Linux: http://*CPU*limit.sourceforge.net. Sept. 15 2007.

[13] C. Stewart and K. Shen. Performance modeling and system management for multi-component online services. Proc. of the 2nd USENIX NSDI, 2005.

[14] P. Dinda. Online Prediction of the Running Time of Tasks. Cluster Computing, 5(3), 2002.

[15] P. A. Dinda and D. R. O'Hallaron. Host load prediction using linear models. Cluster Computing, 3(4):265–280, 2000.

[16] Y. Zhang, W. Sun, and Y. Inoguchi. Predicting Running Time of Grid Tasks based on CPU Load Predictions. In Proceedings of the IEEE/ACM International Conference on Grid Computing 2006.

[17] R. Wolski, N. T. Spring, and J. Hayes. Predicting the CPU Availability of Time-shared Unix Systems on the Computational Grid. Cluster Computing, 3(4), 2000.

[18] R. Gibbons. A historical application profiler for use by parallel schedulers. In IPPS '97: Proceedings of the Job Scheduling Strategies for Parallel Processing1997.

[19] W. Smith, I. Foster, and V. Taylor. Predicting Application Run Times Using Historical Information. Lecture Notes in Computer Science, 1459:122–135, 1998.

[20] A. Goyeneche, G. Terstyanszky, T. Delaitre, S. Winter, Improving Grid computing performance prediction using weighted templates Conf. Proc. of the UK e-Science 2007 All Hands Meeting, 2007.

[21] Francesc Guim, Ivan Rodero, Julita Corbalan, A. Goyeneche. The Grid Backfilling: a Multi-Site Scheduling Architecture with Data Mining Prediction Techniques. CoreGrid Workshop in Grid Middleware 2007.

[22] S. A. Jarvis et al. Performance prediction and its use in parallel and distributed computing systems. Future Gener. Comput. Syst., 22(7):745–754, 2006.

[23] R. Badia, F. Escale, E. Gabriel , J. Gimenez, R. Keller, J. Labarta, M. S. Müller, Perf. Prediction in a Grid Environment, European Across Grids Conf., 2003.