Online Prediction of the Running Time of Tasks: Summary

Peter A. Dinda
pdinda@cs.northwestern.edu
Department of Computer Science
Northwestern University

1. INTRODUCTION

This paper describes a system, the Running Time Advisor (or RTA), that can predict, at run-time, the running time of a compute-bound task on a shared host running a variant of the Unix operating system. Such predictions are valuable for scheduling the soft real-time tasks of distributed interactive applications such as scientific visualizations. To characterize the variability inherent to distributed systems and to the process of prediction, the RTA predicts a task's running time as a confidence interval computed to the application's requested confidence level. Confidence intervals provide a simple abstraction to the application, but still provide sufficient information to enable valid statistical reasoning in the scheduling process.

Figure 1 shows the structure of the Running Time Advisor (or RTA) system, the broader context of which it is a part, and the queries and responses at each level. The RTA's response is computed from host load predictions, a topic we have thoroughly studied in previous papers [1, 4]. We have found that host load, specifically the Digital Unix 5 second load average sampled at 1 Hz, can be usefully predicted to a 30 second horizon using simple AR(16) models. We have implemented an extremely low overhead online host load prediction system, based on a general purpose toolkit [3] that we have made publicly available. [1]

Due to the limited space available, we give only a brief overview of the RTA algorithm here. A full discussion of the RTA (and the real-time scheduler noted in the figure) is available elsewhere [2, Chapters 5 and 6]. The RTA's algorithm simply relates the nominal time (CPU demand) of the task and the running time of the task to the average of the host load signal over the task's running time. We replace the signal in this relationship with the signal prediction and use the covariance matrix of the prediction errors to estimate the confidence interval using a normality assumption. In addition to AR(16), we also explored the use of LAST (last value is predicted) and MEAN (long-term average is predicted) predictors, computing covariance matrices for these predictors appropriately. In addition, we discount the predicted load signal to model the priority boost that processes see when they complete an I/O operation.

We evaluated how well the actual RTA system works in practice using a randomized approach. The evaluation used a real environment where the background load on a host was supplied by host load trace playback [5]. Host load trace playback lets us reconstruct a realistic repeatable workload using a host load trace collected on a real machine. We used traces from 39 different machines. The traces are described in detail in a previous paper [1] and are representative of production and research clusters, application servers, and desktops. We have made the traces and the playback tool publically available. [2]

For each trace, we ran approximately 3000 testcases, which consisted of tasks with nominal times randomly selected from 0.1 to 10 seconds randomly arriving 5 to 15 seconds apart. We used the following the following two metrics: (1) coverage, the fraction of tasks which

[1] http://www.cs.nwu.edu/∼pdinda/RPS.html
[2] http://www.cs.nwu.edu/∼pdinda/LoadTraces

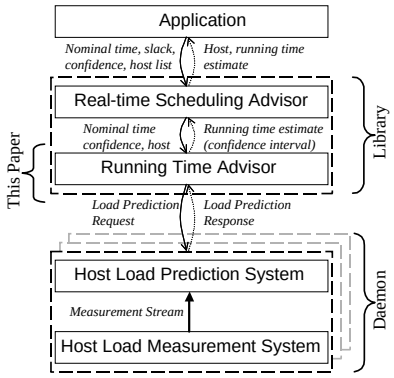


Figure 1: Running Time Advisor (RTA) system and context.

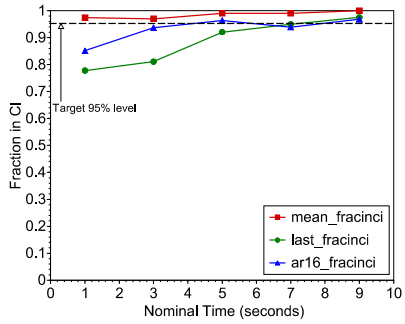
complete with their predicted confidence intervals; and (2) span, the average width of the confidence interval width in seconds. We used a target confidence level of 95%. The main conclusion is that the RTA and its algorithm can indeed predict the running time of tasks in a useful and effective way.

2. EVALUATION RESULTS

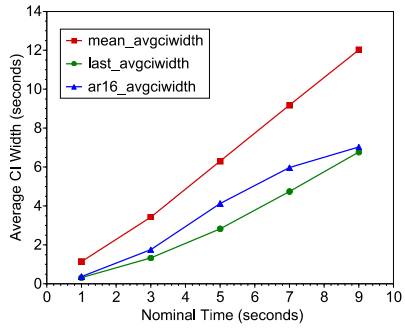
Using the AR(16) predictor, we found that there were only five traces (out of 39) in which fewer than 90% of the tasks completed in their computed confidence intervals and only one host where fewer than 85% were within their intervals. Furthermore, we found that it is generally best to use the most aggressive host load predictor, AR(16). On hosts with high load, AR(16) is able to produce significantly better coverage by estimating wider spans. On hosts with low load, AR(16) can achieve the target coverage with much smaller spans. Performance generally improves as nominal time is increased. We saw five different classes of behavior. In the following, we'll illustrate the two of those and then summarize our overall results.

Class I: This class, which we also call the "typical low load host" class represents the most common behavior by far that we have encountered. The class consists of 29 of the 39 hosts (76%). A representative of class I is plotted in Figure 2. The main characteristics of the class are the following. The coverage is only slightly dependent on the nominal time, increasing slightly for all predictors as the nominal time increases. The MEAN predictor typically has almost 100% coverage and is closely followed by the AR(16) and then the LAST predictor. The LAST and AR(16) predictors have significantly narrower spans than the MEAN predictor, with AR(16) producing slightly wider spans than LAST.

We believe that the AR(16) is the best predictor for this most common class of host. The coverage is nearly as good as MEAN and is typically near the target 95% point, while LAST tends to lag behind, especially for smaller tasks. Furthermore, the span of AR(16) is typically half that of MEAN and only slightly wider than LAST. In most hosts, then, a better predictor produces much narrower accurate confidence intervals.

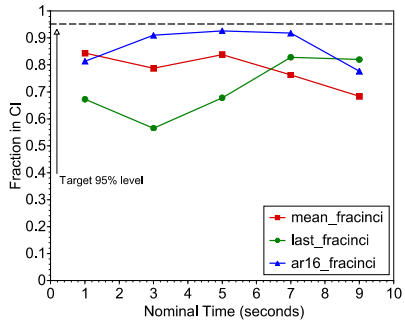


(a) Coverage

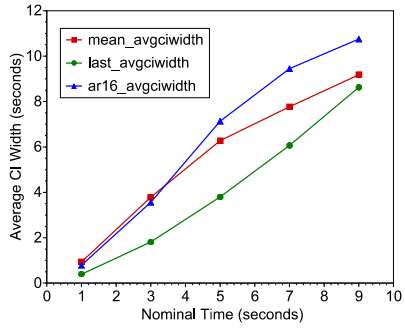


(b) Span

Figure 2: Coverage and Span on Class I hosts



(a) Coverage



(b) Span

Figure 3: Coverage and Span on Class V hosts

Class V: This class, which we also refer to as the "high load 3" class, consists of a single host (2.5%). Figure 3 plots the performance of the predictors on that host using the same methodology as before. In terms of coverage, AR(16) is clearly the winner here, especially for medium sized tasks. It achieves its reasonable coverage (the goal is 95%) by computing slightly larger confidence intervals than MEAN. LAST computes confidence intervals that are far too small, resulting in abysmal coverage.

Generalized results: In the following, we summarize our conclusions based on the class-by-class and other analysis.
(1) The RTA works: with almost every load trace in our study, the coverage of either the AR(16) or LAST predictor is very close to the target 95% coverage.
(2) LAST and AR(16) produce better coverage on heavily loaded hosts: The LAST and AR(16) predictors are better able to "understand" such hosts and compute appropriately wider confidence intervals compared to MEAN.
(3) LAST and AR(16) produce better spans on lightly loaded hosts: For those hosts which have lower load and variability, the LAST and AR(16) predictors produce significantly narrower confidence intervals than MEAN while still capturing an appropriate number of tasks within their computed confidence intervals.
(4) AR(16) performs better than LAST: The confidence intervals computed using AR(16) generally include more of their tasks than those computed using LAST. Using the AR(16) predictor, only five of the traces are at less than 90% and only one less than 85%. Using LAST, 9 are less than 90%, while four are less than 85%. This gain is due to AR(16) predictors producing wider confidence intervals on heavily loaded hosts. There is also a corresponding performance gain on lightly loaded hosts, where AR(16) produces narrower confidence intervals than LAST because it is able to appropriately relax its coverage even more than LAST.
(5) Performance is slightly dependent on the nominal time: For very small tasks, especially those on the order of the measurement period (1 second) or smaller, coverage is worse than for larger tasks. For very long tasks, we see a decline in performance on some hosts. Generally, then, as the nominal time increases, coverage improves slightly. Not surprisingly, spans grow with nominal times.

3. CONCLUSION AND FUTURE WORK

We provided here a high-level description of the Running Time Advisor, a system for predicting the running time of compute-bound tasks, and summarized its performance evaluation. We are currently working on a similar system to predict communication times. The goal is to be able to predict, again as a confidence interval, how long it will take to transfer a given number of bytes between two hosts. We are currently exploring the use of wavelet-based methods to represent, compress, and predict resource signals such as host load. We also plan to extend our prediction work to the application, providing predictions of resource demand as well as resource supply.

4. REFERENCES

[1] P. A. Dinda. The statistical properties of host load. *Scientific Programming*, 7(3,4), 1999. A version of this paper is also available as CMU Technical Report CMU-CS-TR-98-175. A much earlier version appears in LCR '98 and as CMU-CS-TR-98-143.

[2] P. A. Dinda. *Resource Signal Prediction and Its Application to Real-time Scheduling Advisors*. PhD thesis, School of Computer Science, Carnegie Mellon University, May 2000. Available as Carnegie Mellon University Computer Science Department Technical Report CMU-CS-00-131.

[3] P. A. Dinda and D. R. O'Hallaron. An extensible toolkit for resource prediction in distributed systems. Technical Report CMU-CS-99-138, School of Computer Science, Carnegie Mellon University, July 1999.

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

[5] P. A. Dinda and D. R. O'Hallaron. Realistic CPU workloads through host load trace playback. In *Proc. of 5th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR2000)*, May 2000. To appear.