IEEE distributed systems ONLINE
Expert-authored articles and resources

IEEE DISTRIBUTED SYSTEMS ONLINE 1541-4922 © 2004 Published by the IEEE Computer Society
Vol. 5, No. 4; April 2004

Estimating Computation Times of Data-Intensive Applications

Shonali Krishnaswamy, *Monash University*
Seng Wai Loke, *Monash University*
Arkady Zaslavsky, *Monash University*

Accurately estimating the computation times of data-intensive applications is a key component of successful scheduling. The authors show that a rough-sets-based approach can facilitate better estimation and improve overall performance.

A key aspect of scheduling data-intensive applications is the ability to accurately estimate their processing or computation times. Such techniques can improve the performance of scheduling algorithms and help estimate queue times in high-performance computing environments.[1,2]

Application runtime prediction algorithms[1-4] operate on the principle that applications with similar characteristics have similar runtimes. Thus, we maintain a history of applications that have executed along with their respective runtimes. To estimate a given application's runtime, we identify similar applications in the history and then compute a statistical estimate (such as the mean and linear regression) of their runtimes. We use this as the predicted runtime.

The fundamental problem with this approach is the definition of *similarity*; diverse views exist on the criteria that make two applications similar. For instance, we can say that two applications are similar because the same user on the same machine submitted them or because they have the same application name and are required to operate on the same-size data. Thus, we must develop techniques that can effectively identify similar applications.[1] Such techniques must be able to accurately choose applications' attributes that best determine similarity. The obvious test for such techniques is to measure the prediction accuracy of the estimates we obtain by computing a statistical estimate of the runtimes of the applications we identified as similar. Thus, the closer the predicted runtime is to the actual runtime, the better is the technique's prediction accuracy. Having identified a similarity template, the next

step is to estimate the applications' runtime based on previous, similar applications. We can use several statistical measures to compute the prediction, including measures of central tendency such as the mean and linear regression [1]

In this article, we present a holistic approach to estimation that uses *rough sets* theory to determine a similarity template and then compute a runtime estimate using identified similar applications. We tested the technique in two real-life data-intensive applications: data mining and high-performance computing. (See the "Related Work" sidebar for other work in this area.)

ROUGH-SETS-BASED ESTIMATION

The objective of similarity templates in application runtime estimation is to identify a set of characteristics on the basis of which we can compare applications. We could try identical matching—that is, if *n* characteristics are recorded in the history, two applications are similar if they are identical with respect to all *n* properties. However, this considerably limits our ability to find similar applications because not all recorded properties are necessarily relevant in determining the runtime. Such an approach could also lead to errors, as applications that have important similarities might be considered dissimilar even if they differed in a characteristic that had little bearing on the runtime. This has been the main reason for previous efforts[2-4] that use subsets of the properties recorded in the history.

Rough sets theory[5]—introduced by Zdislaw Pawla[6,7] as a mathematical tool to deal with uncertainty in data—provides us with a sound theoretical basis to determine the properties that define similarity. The history represents an information system (as Figure 1 shows) in which the objects are the previous applications whose runtimes (and other properties) have been recorded. The attributes in the information system are these applications' properties. The decision attribute is the application runtime, and the other recorded properties constitute the condition attributes. This history model intuitively facilitates reasoning about the recorded properties so as to identify the dependency between the recorded attributes and the runtime. Thus, we can concretize similarity in terms of the condition attributes that are relevant and significant in determining the runtime. Thus, the set of attributes that have a strong dependency relation with the runtime can form a good similarity template. Rough sets operate entirely on the basis of the data that is available in the history and require no external additional information, which is particularly important because the lack of such information (beyond common sense and intuition) was the bane of manual similarity-template selection techniques. Having cast the problem of application runtime as a rough information system, we can examine the fundamental concepts that are applicable in determining the similarity template.

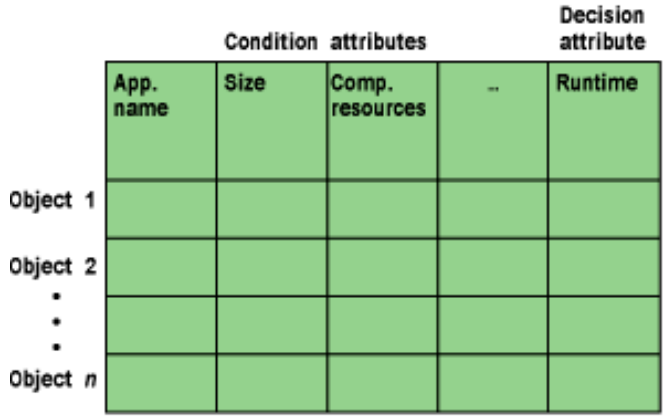

Figure 1. A task history modeled as a rough information system.

Degree of dependency

We can state the problem of identifying a similarity template as identifying a set of condition attributes in the history that have a strong degree of dependency with the runtime. This measure (that is, the degree of dependency) computes the extent of the dependency between a set of condition attributes and the decision attribute. Therefore, it's an important aspect of using rough sets for identifying similarity templates.

The degree of dependency varies in the range [0, 1], with 1 indicating total dependency (or a strong relationship between the condition and decision attributes) and 0 indicating independence between the attribute sets. Obviously, the degree of dependency varies depending on the set of condition attributes considered. Thus, our aim is to find the condition attributes in the information system that clearly relate to the decision attributes.

Significance of attributes

The similarity template consists of a set of properties that are important for determining the runtime. An attribute's *significance* lets us compute the extent to which an attribute affects the dependency between a set of condition attributes and a decision attribute. This in turn lets us identify and consequently eliminate attributes that don't affect the runtime and, therefore, shouldn't be the basis for comparing applications for similarity.

An attribute's significance, like the degree of dependency, has a value in the range [0, 1]. The larger the value, the greater the significance. Basically, to measure an attribute's significance, we

- Compute the dependency between a set of condition attributes and the decision attribute
- Add the attribute for which the significance must be determined to the set of condition attributes
- Recalculate the degree of dependency between the new set of condition attributes and the decision attributes. The difference in the two calculations indicates the impact of the attribute under consideration. If the dependency increases, the attribute is significant, and if the dependency remains unchanged, the attribute is superfluous.

Reduct

A *reduct* consists of the minimal set of condition attributes that have the same discerning power as the entire information system. We eliminate all superfluous attributes from a reduct. A similarity template should consist of the most important set of attributes that determine the runtime without any superfluous attributes. In other words, the similarity template is equivalent to a reduct that includes the most significant attributes.

Rough-sets theory has highly suitable and appropriate constructs for identifying the properties that best define similarity for estimating application runtime. A similarity template must include attributes that significantly affect the runtime and eliminate those that don't. This ensures that the criteria with which we compare applications for similarity have a significant bearing on determining runtime. Consequently, applications that have the same characteristics with respect to these criteria will have similar runtimes.

In computing a reduct for use as a similarity template in application runtime estimation, we require the reduct to consist of attributes that are significant with respect to the runtime. For this purpose, the algorithm we use operates as follows (see Figure 2). Initially, it computes the information system's core. Then, it computes the significance of each condition attribute that isn't part of the core. It then calculates the initial degree of dependence between all the condition attributes (namely the application characteristics) and the decision attribute (namely the recorded computation and runtime). Finally, the algorithm computes reducts by iteratively removing attributes that have the least significance in determining the dependence between the condition and decision attributes. After each removal, the algorithm recomputes the degree of dependency between the remaining condition attributes and decision attribute; if the dependency remains the same, the attribute is eliminated, and if the dependency changes, the algorithm deems the attribute relevant and reintroduces it to the set of attributes that constitute the reduct. This process repeats for all condition attributes that aren't part of the core. Thus, at the final step, we find a reduct that consists of all the most significant attributes and use it as the similarity template for estimating application runtimes. This approach is similar to Hu's technique,[8] which focuses on user-specified reducts and incorporates both a forward- and backward-chaining approach to compute reducts. A distinguishing characteristic of our approach is that we apply only a backward-chaining process to compute the reducts.

1. Let A = $\{a_1, a_2, \ldots, a_n\}$ be the set of condition attributes and D = $\{d_1, d_2, \ldots, d_\lambda\}$ be the set of decision attributes.

2. Let C = $\{c_1, c_2, \ldots, c_m\}$ be the D-Core, $c_i \in$ A and $m \leq n$

3. Reduct = C

4. A1 = A - Reduct

5. Compute the Significances of the Attributes (SGF) in A1 and sort them in descending order of the significance:

$A1_{sort} = \{a_{sort1}, a_{sort2}, \ldots, a_{sortn}\}$

6. For i = 1 to $|A1_{sort}|$

K(Reduct, D) = K(Reduct, D) − SGF(a_{sorti})

If K(Reduct, D) = K(A, D)

Reduct = Reduct + a_{sorti}

Exit Program /* Reduct Found */

End If

K(Reduct, D) = K(Reduct, D)- SGF(a_{sorti})

End For

/* Begin Backward Chaining Process */

7. Reduct = A

8. For $i = |A1_{SORT}|$ to 1

Reduct = Reduct − a_{sorti}

(note: $a_{sorti} \in A1_{sort}$ and therefore $a_{sorti} \notin$ C)

If K(Reduct, D) not equal to K(A, D)

Reduct = Reduct + a_{sorti}

End If

End For

Figure 2. Algorithm for generating the reduct we use as a similarity template.

Estimation algorithm

Let's now look at the estimation algorithm as a whole. Its input is a recorded history of application characteristics

collected over time, specifically including actual recorded runtimes, and a task T with known parameters whose runtime we wish to estimate.

Step 1. Partition the history into decision and condition attributes. The recorded runtime is the decision attribute, and the other recorded characteristics are the condition attributes. The approach is to record a comprehensive history of all possible statistics with respect to an application because identifying the attributes that determine the runtime isn't always possible.

Step 2. Apply the rough-sets algorithm to the history and identify the similarity template.

Step 3. Combine the current task T with the history H to form a current history HT .

Step 4. Determine from HT the equivalence classes with respect to the identified similarity templates. This implies grouping into classes previous tasks in the history that are identical with respect to the similarity template. Because the similarity template generated using rough sets is a reduct, this leads to the equivalence classes consisting of previous tasks that are identical with respect to the characteristics that have the most significant bearing on the runtime. In this case, rough sets provide a basis for identifying the similarity template and finding previous tasks that match the current task by the intuitive use of equivalence classes. Thus, we integrate the process of matching the current task with previous tasks in the history into the overall process of estimating application runtime.

Step 5. Identify the equivalence class EQ to which T belongs.

Step 6. Compute t he mean of the runtimes of the objects: $EQ \cap H$.

The formal algorithm. Figure 3 offers a formal view of the estimation algorithm. As we explained previously, the entire process of identifying similarity templates and matching tasks to similar tasks is based on rough-sets theory, thereby providing an appropriate solution with a strong mathematical underpinning.

Input: History of tasks = H , Current task for which the runtime has to be estimated = T.

1. Partition H such that the runtime is the decision attribute and all the other recorded characteristics are the condition attributes.
2. Apply the rough sets algorithm to the history and generate a similarity template ST.
3. Let $H_T = H + T$, where H and T are union compatible.
4. Compute equivalence classes of H_T with respect to ST.
5. Identify the equivalence class EQ_T to which T belongs.
6. Compute the mean of the recorded runtimes EST in H for all objects in EQ_T (note: $T \notin H$).

Output: Estimated runtime EST.

Figure 3. Application runtime estimation using rough sets.

EXPERIMENTAL SCENARIOS

We applied our rough-sets approach in two data-intensive applications. First, we analyzed large datasets in a data-mining application, estimating the computation times of data-mining tasks. Second, we used data from a high-performance computational facility where both data- and processor-intensive tasks are executed. In this scenario, a need exists to schedule such tasks and assign them servers. These diverse scenarios establish the generality and validity of our approach for different domains.

In both experiments, we differentiated the test case from the history's records by removing the runtime information. Thus, a test case consists of all the information specified except the recorded runtime. The runtime information recorded in the test case was the task's actual runtime. The idea was to determine an estimated runtime using our prediction technique and compare it with the task's actual runtime.

Data mining

We compiled a history of data-mining tasks by running several data-mining algorithms on a network of distributed machines and recording information about the tasks and environment. We executed several runs of data-mining jobs by varying the jobs' parameters such as the mining algorithm, the data sets, the sizes of the data sets, and the machines on which the tasks were run. The algorithms we used were from the Weka package of data-mining algorithms.[9] We generated several data sets of sizes varying from 1 to 20 Mbytes.

We executed the data-mining jobs on three distributed machines with different physical configurations and operating systems: a Pentium III running Windows 2000 with an 833-MHz processor and 512 Mbytes of memory; a Pentium II running Windows 2000 with a 433-MHz processor and 128 Mbytes of memory; and a Sun Sparc station running Sun OS 5.8 with a 444-Mhz processor and 256 Mbytes of memory. For each data-mining job, we recorded the following information in the history: the algorithm, file name, file size, operating system, operating system version, IP address of the local host on which the job was run, processor speed, amount of memory, and start and end times. We used histories with 100 and 150 records, and as before, each experimental run consisted of 20 tests. Seven steps are involved:

Input: History and test case

1. Compute a reduct using the history as the data.

2. Use the reduct as a similarity template—that is, use the reduct's attributes to compare the test case with similar past jobs that were recorded in the history.

3. Add the test case to the history and compute equivalence classes with respect to the similarity template—that is, group all tasks that are indiscernible with respect to the similarity template into equivalence classes.

4. Determine the equivalence class containing the test case, which consists of previous jobs in the history that are similar to the current test case.

5. Compute the mean of the runtimes of similar jobs as the estimated runtime. (Smith showed that the mean performs well as an estimator. It gives us a good basis for comparison. 2)

Output: Estimated application runtime for the test case.

In our experiment, the mean error was 1.02 minutes, and the mean error as a percentage of the actual runtimes was 26.4 percent . This shows that we accurately estimated the runtime for data-mining tasks. The reduct that our algorithm selected as a similarity template included the algorithm, file size, dimensionality, and available memory attributes. Figure 4 illustrates the actual and estimated runtimes from one of our experimental runs.

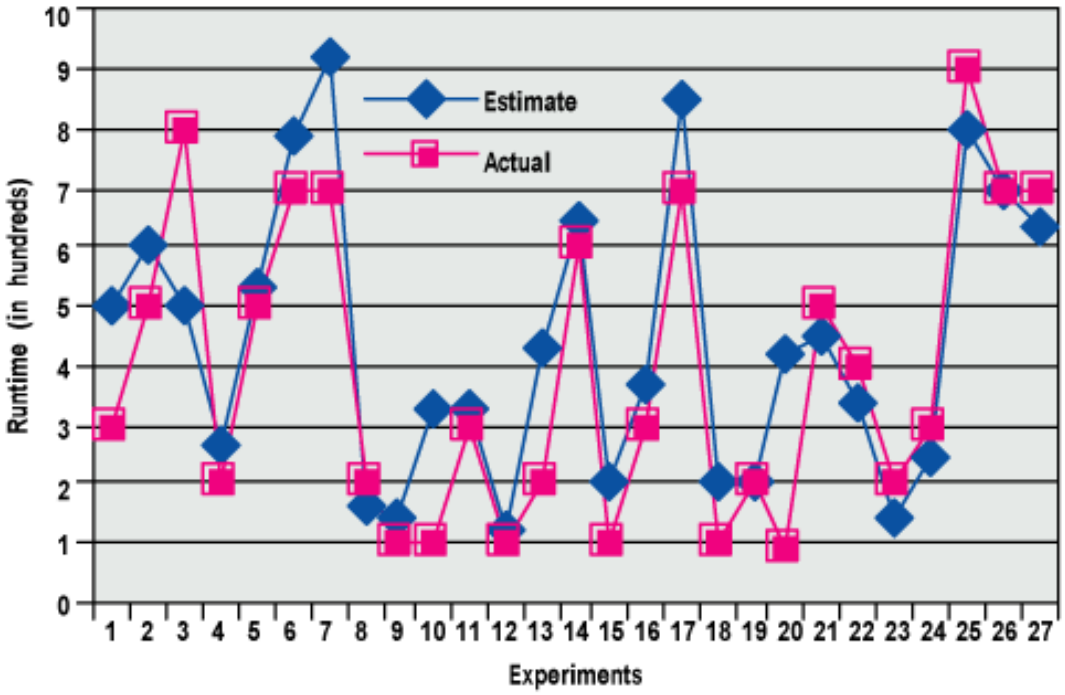

Figure 4. Actual versus estimated computation times from our data-mining experiment.

High-performance computing

In a supercomputing facility that hosts high-performance servers, users can submit various applications, data- or processor-intensive (or both). Users can supply their own software and data or use the locally available software on their respective data. Thus, the type of applications executed can vary widely and, consequently, so can the respective applications' computation times.

To apply our approach in such an environment, we used two data sets that represent parallel-computer workloads that Allen Downey collected at the San Diego Super Computing Supercomputer Center in 1995 and 1996. These two data sets, SDSC95 and SDSC96, have the following information recorded for each job: account name; login name; partition to which the job was allocated; the number of nodes for the job; the job type (batch or interactive); the job status (successful or not); the number of requested CPU hours; the name of the queue to which the job was allocated; the rate of charge for CPU hours and idle hours; and the task's duration in terms of when it was submitted, started, and completed. Thus, the data sets provide us with information that is collected at a higher level of abstraction than the data-mining case that was specific to a particular class of applications. In this example, we have coarser-grained data to test our algorithm.

For each experimental run, we took two samples from the data set: a history and a set of test cases. We tested our algorithm by varying the size of the history from 100 to 1,000. For each run, we used 25 to 30 test cases.

For each data set (SDSC95 and SDSC96), we conducted 10 experimental runs. For each test case, we recorded the reduct, estimated runtime, and actual runtime. For each case, we computed the error and used this to determine the mean error for the different data sets—that is, the mean variation between the actual and estimated runtimes. Because the mean error is the mean of the difference between the actual and estimated values, a lower mean error indicates better prediction accuracy. Figure 5 presents the best, worst, and average mean errors obtained from the 280 test runs per data set. Although the best and worst cases represent the end points in the distribution, we believe the average error is the most representative of the estimation accuracy of the rough-sets approach.

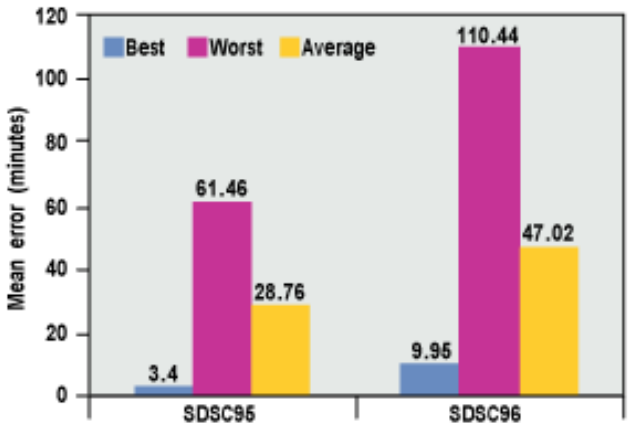

Figure 5. Mean error in estimating computation times.

Another measure we used for comparison is the mean error as a percentage of the mean runtimes (see Table 1). In this case, the rough-sets approach performs exceedingly well with the SDSC95 data set but falls short on the SDSC96 data set. This is principally because we had high mean errors for two experimental runs, which involved applications with long runtimes.

Table 1. Mean error as a percentage of mean runtimes.

	SDSC95 (%)	SDSC96 (%)
Best	0.94	72.59
Average	29.59	126.17
Worst	10.99	85.50

Rough sets operates only on the basis of the data, and on examining the data, we discovered that the applications that resulted in large errors were due to the fact that there were two jobs in the history that were identical to the test case (not merely with respect to the similarity template but to all recorded characteristics) but had completely different runtimes. In general, the algorithm works on the basis of what has been recorded, and in this particular

case, the recorded characteristics didn't sufficiently distinguish the jobs. Our experiments showed that a rough-sets-based approach has good results for estimating computation times in scenarios where application-specific characteristics aren't available at a fine level of granularity.

CONCLUSION

In the future, we plan to apply the estimates we obtain to develop scheduling algorithms for data-intensive applications. The estimation techniques also have potential application for estimating quality-of-service levels in data-centric grid environments and data-intensive Web services. We are currently investigating these areas of application.

Acknowledgments

We thank Allen Downey for making the San Diego Super Computing Supercomputer Center workloads publicly available and for his permission to use them in our experiments. We thank the Australian Research Council for funding this research.

References

1. W. Smith , I. Foster, and V. Taylor , "Predicting Application Runtimes Using Historical Information,"*Job Scheduling Strategies for Parallel Processing: IPPS/SPDP'98 Workshop ,* LNCS 1459, Springer-Verlag, 1998,pp. 122-142.

2. W. Smith , V. Taylor, and I. Foster , "Using Runtime Predictions to Estimate Queue Wait Times and Improve Scheduler Performance,"*Job Scheduling Strategies for Parallel Processing ,* LNCS 1659, D.G. Feitelson and L. Rudolph, eds., Springer-Verlag, 1999,pp. 202-229.

3. A.B. Downey , "Predicting Queue Times on Space-Sharing Parallel Computers,"*Proc. 11th Int'l Parallel Processing Symp.* (IPPS 97), IEEE CS Press, 1997, pp. 209-218; www.sdsc.edu/~downey/predicting.

4. R. Gibbons , "A Historical Application Profiler for Use by Parallel Schedulers,"*Job Scheduling Strategies for Parallel Processing ,* LNCS 1291, Springer-Verlag, 1997,pp. 58-77.

5. J. Komorowski , et al., "Rough Sets: A Tutorial,"*Rough-Fuzzy Hybridization: A New Trend in Decision Making ,* S.K. Pal and A. Skowron, eds., Springer-Verlag, 1998, pp. 3-98.

6. Z. Pawlak , "Rough Sets,"*Int'l J. Computer and Information Sciences,* no. 11, 1982,pp. 341 - 356.

7. Z. Pawlak , *Rough Sets: Theoretical Aspects of Reasoning about Data ,* Kluwer Academic Publishers, 1992.

8. X. Hu , *Knowledge Discovery in Databases: An Attribute-Oriented Rough Sets Approach ,* doctoral dissertation, Univ. of Regina, *Canada,*1995.

9. I.H. Witten and F. Eibe , *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations ,* Morgan Kauffman,

Shonali Krishnaswamy is a research fellow at the School of Computer Science and Software Engineering at Monash University. Her research interests include service-oriented computing, distributed and ubiquitous data mining, software agents, and rough sets. She received her PhD in computer science from Monash University. Contact her at 900 Dandenong Rd., Monash Univ., Caulfield East, Victoria -3145, Australia; Shonali.Krishnaswamy@infotech.monash.edu.au.

Seng Wai Loke is a research fellow in the Faculty of Information Technology at the School of Computer Science and Software Engineering at Monash University. His research interests include intelligent and mobile agents, service-oriented computing, and pervasive computing. He received his PhD in computer science from the University of Mlebourne. Contact him at 900 Dandenong Rd., Monash Univ., Caulfield East, Victoria -3145, Australia; Seng.Loke@infotech.monash.edu.au.

Arkady Zaslavsky is an associate professor at the School of Computer Science and Software Engineering at Monash University. His research interests include mobile and pervasive computing, distributed and mobile objects and agents, wireless networks, distributed computing and databases, and mobile commerce. He received his PhD in computer science from the Moscow Institute for Control Sciences, USSR Academy of Science. Contact him at 900 Dandenong Rd., Monash Univ., Caulfield East, Victoria -3145, Australia; Arkady.Zaslavsky@infotech.monash.edu.au.

Related Work

Early work in this area[1,2] proposed using *similarity templates* of application characteristics to identify similar tasks in a history. A similarity template is a set of attributes that we use to compare applications in order to determine if they are similar. Thus, for histories recorded from parallel-computer workloads, one set of researchers selected the queue name as the characteristic to determine similarity.[1] They considered that applications assigned to the same queue were similar. In other work,[2] researchers used several templates for the same history, including user, application name, number of nodes, and age.

Warren Smith, Ian Foster, and Valerie Taylor[3] proposed that manually selecting similarity templates had the following limitations:

- Identifying the characteristics that best determine similarity isn't always possible.
- It's not generic: although a particular set of characteristics might be appropriate for one domain, it's not always applicable to other domains.

They proposed that definition and search for templates must be automated, and they used genetic algorithms and greedy-search techniques.[3] They were able to obtain improved prediction accuracy using these techniques over manually selecting similarity templates. The greedy search basically identifies a set of similarity templates with a varying number of attributes (such as similarity template with two attributes, three attributes, and so on). Their experimental evaluation showed that numerous attributes in the template lead to higher accuracy in the estimates. The genetic algorithm learns the best similarity template from the history available. The genetic algorithm provided better accuracy than the greedy-search technique.

We previously developed a rough-sets-based technique to address the problem of automatically selecting characteristics that best define similarity.[4] In this article, we extend our rough-sets approach to the entire application runtime estimation process.

References

1. A.B. Downey , "Predicting Queue Times on Space-Sharing Parallel Computers,"*Proc. 11th Int'l Parallel Processing Symp.* (IPPS 97), IEEE CS Press, 1997,pp. 209-218; www.sdsc.edu/~downey/predicting.
2. R. Gibbons , "A Historical Application Profiler for Use by Parallel Schedulers,"*Job Scheduling Strategies for Parallel Processing ,* LNCS 1291, Springer-Verlag, 1997,pp. 58-77.
3. W. Smith , I. Foster, and V. Taylor , "Predicting Application Runtimes Using Historical Information,"*Job Scheduling Strategies for Parallel Processing: IPPS/SPDP'98 Workshop ,* LNCS 1459, Springer-Verlag, 1998,pp. 122-142.
4. S. Krishnaswamy , A. Zaslasvky, and S.W. Loke , "Predicting Application Runtimes Using Rough Sets,"*Proc. 9th Int'l Conf. Information Processing and Management of Uncertainty in Knowledge-Based Systems* (IPMU 2002), IEEE Press, 2002,pp. 455-462.