Predicting Runtimes for Production Parallel Jobs

A Project Report
Submitted in partial fulfilment of the
requirements for the Degree of
Master of Technology
in
Computational Science

by

Gowthami Manogna Gottipati

Supercomputer Education and Research Centre
Indian Institute of Science
BANGALORE – 560 012

JULY 2012

Acknowledgements

I would like to express my gratitude to my advisor Dr. Sathish Vadhiyar for his guidance, support and encouragement. His wise counsel had made my research experience enriching and rewarding. I would also like to thank Prof. Govindarajan, Dr. Atanu Mohanty, Dr. Virender Singh, Dr. Shirish Shevade, Dr. Sathish Govindarajan and all others who helped me gain knowledge through various courses. I would like to thank the SERC department for providing me with the computing facilities.

I take this opportunity to thank my lab members and batchmates for their help and support through out the year. I would like to thank Sharat for his support and encouragement during my stay in IISc. Thanks are also due to my friends Shalini and Maria for making life in IISc a happy experience.

I am indebted to my parents and my sister for being my pillars of strength.

Abstract

The aim of this project is to develop strategies to predict the execution times for parallel jobs that are submitted to production batch queues using historical information. We used an existing technique that chooses jobs from the history relevant to a particular job and uses the history to make a runtime prediction. We developed filtering techniques in which the jobs in the history of a target job are divided into classes based on their run times. These classes are then filtered on the basis of the average euclidean distance from the target job and the size of the class. We also employed and compared various machine learning techniques for predictions using the filtered history. The combination of filtering and machine learning techniques provided 14% improvement over the existing technique. Of the various machine learning techniques, Lasso Regression provided the best results. We also analyzed the various reasons for large prediction errors in different cases.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Production supercomputers are batch systems in which jobs are submitted to batch queues. Batch jobs contain all their input data in scripts or command-line parameters and the output data is put in a set of files. These jobs are put in different queues based on their priorities. Scheduling algorithms are used to pick jobs from the queues and allocate the resources required by the job. Each job has certain characteristics like the queue in which it is present, the user who submitted it, its load leveler script, executable, arguments, number of nodes requested by it, maximum runtime or the user estimated runtime, submission time, start time, execution time etc.

The runtime estimates are very useful to the users of the batch systems and to the batch schedulers. The runtime estimates are used to predict how long a job has to wait before execution. Based on this wait time estimate, users can choose among the various batch systems available to them. The Batch Schedulers use the run time and queue wait time estimates to automatically pick jobs from the queues and allocate the resources in a way that helps reduce the time complexities associated with the job submission, i.e turn around times, wait times and response times. The use of the run time and queue waiting time estimates by the users and schedulers can lead to a balanced load on the systems and to an efficient use of the resources.

1.1 Scheduling Algorithms

Some widely used scheduling algorithms are First Come First Served, Shortest Job First, Least Work First, Fairshare Scheduling, Priority Escalation etc. Some of these include techniques like Gang Scheduling and Back Filling. The criteria to measure the performance of the scheduling policies are the throughput, turn around time, response time, waiting time and fairness. The work of Etsion et al. [8] shows that most of the batch schedulers assign priorities to jobs based on the estimated runtimes of the jobs and support backfilling.

1.2 Back Filling

When the first job in the queue is waiting for the jobs running at that time to release the nodes, some job from the queue whose node requirements can be met is picked and alloted the free nodes. This is back filling. In conservative back filling, the scheduler picks the job that does'nt delay the start time of the first job in the queue. It picks a job only if its runtime estimate does'nt exceed the time that the first job in the queue has to wait if back filling is not applied. To do this, the scheduler uses runtime estimates of the jobs running at that time and of the jobs that it wants to pick from the queue for back filling. In agressive back filling , the first job in the queue that can be scheduled is picked. This can lead to starvation of some jobs.

Back filling tries to reduce the number of idling nodes at any given time. Since this improves the efficiency of the system, most of the existing batch schedulers employ back filling. The results of Tang et al. [1] show that using more accurate runtime estimates improves the performance of scheduling policies that favor shorter jobs and that higher accuracy of runtime estimates in such policies results in more backfilling.

The estimates used by the schedulers can be the runtime requests submitted by the users along with their jobs. The jobs are terminated by the scheduler when they run beyond the times requested by the user. But the user runtime estimates are shown to have high inaccuracy by Mu'alem et al. [3]. The users tend to overestimate the runtimes to avoid the jobs from being killed when the estimates are exceeded. They also tend to round the estimates to multiples of 5 or 10 minutes. Hence, techniques to predict the runtimes are employed.

1.3 RunTime Prediction

There are several existing strategies for predicting execution times of parallel applications. They predict the runtimes by using analytical models, historical information, application knowledge and compiler information.

The application based models require the application developer to provide detailed knowledge of the applications. This might prevent them from integrating their applications into grids. Some techniques analyze the source code of the applications for the construction of the models and some capture the compiler transformations. The transformations are stored by the compiler automatically [9] [10] [11] [12]. These techniques are time-consuming for large and complex applications. The performance modeling techniques require multiple executions of the job with varying problem sizes and number of nodes. Some of these techniques are intended only for simple parallel kernels that have single phase of uniform computations and communications among processors.

The analytical models cannot be used in predicting runtimes for the batch systems since the history for a kind of jobs is limited and is less organized for batch systems when compared to other parallel environments. Also, the training period for the batch systems is not well defined and training can be done at different points of time to include the jobs that have recently finished execution.

Our plan is to develop a strategy to predict execution times using historical information. The work of Smith et al. [6] [7] predicts runtimes of parallel jobs using historical information. A template set is used for choosing similar jobs from history and it is determined by applying search techniques like genetic search and greedy search. Prediction techniques like mean or linear regression are applied on the runtimes of the jobs in the history to obtain the runtime prediction for each template in the template set and the corresponding confidence intervals are computed. The prediction with the minumum confidence interval value is chosen. Their results show high runtime prediction errors.

1.4 Problem Statement

The existing work in predicting runtimes using historical information, the work of Smith et al. [6] [7] shows high prediction errors. These predictions need to be improved by refining the selection of history for a job and the techniques applied to obtain the estimate.**We developed a comprehensive set of strategies including filtering of similar jobs in history, dynamic selection of prediction methodology, and exploration of machine learning techniques, for predictions of runtimes in production supercomputers**.

The filtering techniques reduce the noise in choosing similar jobs. The jobs in the history of a target job are divided into classes based on their run times. These classes are then filtered on the basis of the average euclidean distance from the target job and the size of the class to reduce the noise in choosing the similar jobs. We also devised a strategy which dynamically decides whether the filtered history or the original Smith's history should be used.

We employed and compared various machine learning techniques for predictions using the history corresponding to the dynamically chosen prediction. The combination of filtering and machine learning techniques provided 14% improvement over the existing technique. Of the various machine learning techniques, Lasso Regression provided the best results. We also analyzed the reasons for large prediction errors in different cases.

1.5 Organization

In Chapter 2, we discussed the literature related to our work. The shortcomings of the already proposed strategies and the reasons why a new strategy is required to approach the given problem statement are discussed. The work of Smith et al. [6], along with which we tried to employ our stragies to improve run time predictions is also discussed.

Chapter 3 describes the methodology proposed by discussing the filtering techniques, the dynamic selection technique and the use of machine learning techniques. It also includes a diagram to explain the flow of the implementation of these techniques.

In Chapter 4, the experimental setup and the results obtained are presented. The details of the traces on which our techniques are applied are given. The results include a table that shows a comparison amongst various machine learning techniques and a table that shows the effect of inclusion of filtering techniques, dynamic selection technique and machine learning techniques in Smith's work.

In Chapter 5, we discussed our observations on high error predictions for each of the traces and show the grouping of high error predictions according to the possible reasons for bad predictions.

Chapter 6 discusses the conclusions of our project and the future work that can be done to extend this work.

Chapter 2

Related Work

This chapter presents the literature related to our work. It also discusses why we chose to work on runtime prediction stratagies that use historical information.

2.1 The effect of Accuracy of RunTime Estimates on Job Scheduling

The work of Mu'alem et al. [3] compares the performance of aggressive backfilling with conservative backfilling and concludes that their relative performance depends on the workload. They study the sensitivity of backfilling to the accuracy of the user runtime estimates. Their results show that accurate runtime estimates are not necesarily the ones that lead to the best schedules, because some amount of inaccuracy in the estimates gives the scheduling algorithms more flexibility. They also show that backfilling works better when users over estimate the runtime by a substantial factor.

The work of Tsafrir et al. [2] presents similar results. They use different values over a range starting at the actual runtime to obtain a runtime estimate and observe the effect it has on backfilling. They find that inaccurate estimates yielded better results than the exact runtimes in backfilling.

The work of Tang et al. [1] analyses the effect of improved accuracy of runtime estimates on job scheduling policies like FCFS backfilling and WFP backfilling. They show that using more accurate runtime estimates improves the performance of scheduling policies that favor shorter jobs. They also explore on the impact of increased accuracy in runtime estimates on various parts of the scheduling algorithm. They show that the runtime estimation accuracy in queuing job priority is more important than that used in the backfilling algorithm. Also, using more accurate estimation except for calculating the backfilling window of the running jobs is good for performance for the majority of the jobs.

They also present a scheme to adjust user runtime estimates in order to achieve better scheduling performance. The ratio of the actual runtime to the user estimate of the runtime is calculated for each of the jobs after their execution. When a new job arrives its adjusting parameter is obtained by taking the 80th percentile value of the value of these ratios from the jobs history. This history can be user based, project based or both. Their results show an increase of 20% in the system performance when applied on real system workloads. Their work considers only the project and user information of the jobs to decide the job history. There are many other parameters like number of nodes, arguments, class, executable, submission time, start time etc which effect the runtimes. The adjustment strategy here uses the 80th percentile value from the job history. This can be extended by considering prediction techniques like interpolation or machine learning techniques like support vector machines or decision trees.

2.2 Existing Stratagies for RunTime Prediction

The approaches to predict application runtimes in parallel environments are usually based on compiler information, application knowledge, analytical models or history.

The application based models [11] [9] [10] analyse the source code of the applications for the construction of the models and for instrumentation of the critical components. The compiler based techniques [12] study the compiler transformations that can not be captured by analysing the source code. This is done by making the compiler capture the transformations automatically. These techniques are time-consuming for large and complex applications.

The work of Sanjay et al. [5] presents a performance modeling technique to predict runtimes. The runtime predicted by modeling the computation, communication and scalability. The computation time and the communication time are obtained from different parameters which capture the properties of the job and the state of the parallel environment. These parameters are obtained by using model functions, executing the job on a single processor for different problem sizes, executing it on 2 processors for different problem sizes, different cpu and network loads and executing it on 2,4,8 processors for different loads. The errors in their runtime predictions are less than 30% for 72% of the jobs and less than 40% for 85% of the jobs.

These techniques cannot be used for the batch systems since the history for a kind of jobs is limited and is less organized for batch systems when compared to other parallel environments. Also, the training period for the batch systems is not well defined and training can be done at different points of time to include the jobs that have recently finished execution.

2.3 Prediction by Smith [6]

The work of Smith et al. [6] presents an algorithm that uses search techniques (greedy and genetic algorithm) to determine the application characteristics that yield the best definition of similarity for the purpose of making predictions of execution times of parallel jobs. The search technique results in a template set that is optimal in terms of the mean error for the training set of jobs. A Template defines a set of job characteristics that are considered while deciding if two jobs are similar. A prediction technique, either mean or linear regression, is applied on the similar jobs chosen by each template to come up with the prediction and the confidence interval for each prediction is obtained. The prediction with the smallest confidence interval is chosen as the runtime prediction.

Their later work [7] uses the runtime predictions made using the technique in [6] to predict the wait times of jobs in the queues for some scheduling algorithms. Their results show that the use of the predicted runtimes and queue waiting times improved the scheduling performance of least work first and back fill scheduling algorithms.

This work yeilds high errors in predictions, the mean error in the runtime prediction is greater than the actual runtimes of most of the jobs. Out of the target jobs that have a high error in runtime prediction, there are jobs that have enough history. The history for such jobs has to be refined. More prediction techniques other than mean or linear regression can be tried on the history chosen.

2.4 Machine Learning Techniques

The work of Matsunaga et al. [13] discusses some machine learning techniques and their suitability for predicting utilization of resources by applications. The techniques considered are k-nn, linear regression, decision tables, radial basis function network and support vector machines. It extends an existing classification algorithm called Predicting Query Runtime(PQR) to regression and also investigates the impact of attributes on prediction accuracy. PQR technique uses decision trees to classify. This technique is extended by selecting the best regression technique for the data at each leaf. The results show that extended PQR has the highest accuracy in the techniques considered as it chooses regression models for different regions of the data at the cost of additional computation during training. The results show that the use of system performance attributes are relevant for execution time prediction whereas application specific attributes were pertinent for all scenarios.

Chapter 3

Methodology

In this project, we have developed filtering techniques to refine the history selected by Smith's method and reduce the noise contributed by dissimilar jobs in history. We also explored the use of machine learning techniques for improving the predictions obtained. The overall methodology is illustrated by a flowchart shown in Figure 3.1.

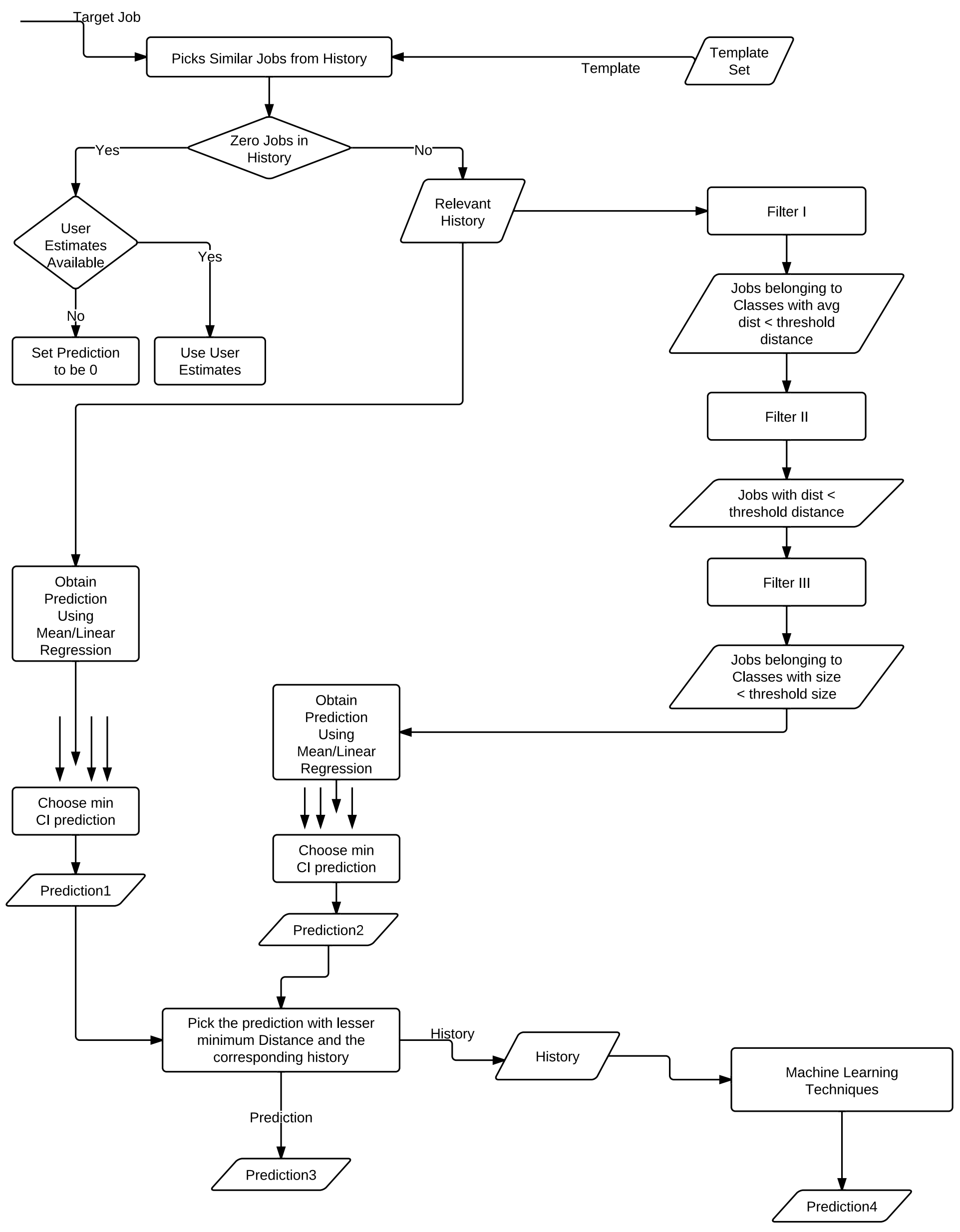
Target Job
Picks Similar Jobs from History
Template
Template Set
Zero Jobs in History
Yes
No
User Estimates Available
Yes
No
Set Prediction to be 0
Use User Estimates
Relevant History
Filter I
Jobs belonging to Classes with avg dist < threshold distance
Filter II
Jobs with dist < threshold distance
Filter III
Jobs belonging to Classes with size < threshold size
Obtain Prediction Using Mean/Linear Regression
Choose min CI prediction
Prediction1
Obtain Prediction Using Mean/Linear Regression
Choose min CI prediction
Prediction2
Pick the prediction with lesser minimum Distance and the corresponding history
History
History
Machine Learning Techniques
Prediction
Prediction3
Prediction4

Figure 3.1: Overall Methodology

3.1 Background: Selection of Similar Jobs in History

In our method, for a given target job, we first obtain jobs in history similar in characteristics to the target job, and then use the runtimes of the similar jobs to predict the runtime of a target job. To obtain similar jobs in history, we use the method by Smith et al. [6]. In this method templates are used to mark the job characteristics that define the similarity of jobs. The jobs from history that have the marked job characteristics same as those of the target job are picked. These jobs are considered to be similar to the target job and their runtimes are used in prediction.

Instead of using a single template for all the jobs, a template set is used. Each template corresponds to a history, a prediction and a confidence interval value for the prediction. When a target job arrives, each of the templates from the set is used to compute the prediction and its confidence interval. The prediction with the minimum confidence interval value will be the runtime estimate for the target job.

Genetic search is used to obtain the best template set for a subset of jobs from the trace that are used as a training set. We used the template set obtained by applying genetic search on the initial 10000 jobs as the training set. This technique starts with a generation containing 10 random template sets, i.e the size of the template set and the templates are randomly initialized. In every generation templates are evaluated on the basis of the mean errors they yeild for the training set and assigned a fitness value. In the selection process the template sets with higher fitness values are more probable of being selected for crossover. The templates are selected in pairs and merged to obtain two new template sets. The two highest fitness value template sets are retained to the next generation. We used the best template set that is given in the work of Smith et al. [6] for those traces they have results for. For the other traces we used 10 iterations of the genetic algorithm to get the best template set.

The prediction technique used is either mean or linear regression. The prediction is either the mean of all the runtimes of the similar jobs or in case of linear regression, it is the value obtained by interpolating the runtimes against the number of nodes.

A confidence interval for a confidence of x% means the actual value of the parameter being estimated lies in an interval of size equal to the confidence interval x% of the times. When the confidence value is fixed, the smaller the confidence interval size the more reliable is that prediction. We computed the 95% confidence interval value for each prediction and chose the prediction with the minimum confidence interval value.

3.2 Refining History using Filtering

All the jobs are put into classes based on their runtimes, as we use the class information while filtering the similar jobs of a target job. Kmeans clustering is used to decide the runtime ranges of classes. The data points fed to Kmean clustering are runtimes. The initial class ranges are obtained by splitting the whole range of runtimes into 10 equal intervals. In every iteration the centroids of classes are computed and the runtimes are put into the class with closest centroid. The number of runtimes changing their class after every iteration is computed and when this number is sufficiently low the procedure is stopped. Thus, each job in the trace is assigned a class based on its runtime.

The similar jobs for a target job belong to various classes. For each of these classes, the average of the euclidean distances of the jobs in that class to the target job is computed. Then a threshold value for the distance is decided based on the minimum of these average distances computed. In our work, we chose this threshold distance as 12 times the minimum distance. The jobs belonging to the classes with average distance greater than the threshold are removed. Of the remaining jobs, the ones with distance from the target job greater than threshold are also removed.

The remaining are jobs belonging to classes which had their initial average distance less than threshold distance and now have some jobs whose distance from target job is less than the threshold distance. The number of jobs in each of these classes is computed and based on the maximum of these numbers a threshold size is decided. We chose the threshold to be 0.3 times the maximum size. Then all the classes with size smaller than the threshold are removed. The jobs left after all the filtration are used to obtain the runtime prediction.

Prediction techniques used by Smith i.e mean and linear regression are now applied on the filtered history to obtain the prediction and confidence interval. The prediction with the minimum confidence interval is chosen. The filtering of history reduced the noise in choosing the similar jobs in some cases and led to a decrease in the mean prediction error.

3.3 Normalization

We normalized the job characteristics to lie between 0 and 1. This is done to ensure that all the characteristics get equal importance when the euclidean distance is computed to measure similarity between jobs. We observed that predictions are better when normalized characteristics are used for computing the distances during filtering and when applying machine learning techniques. Also, the computation of euclidean distance where the dimensions are the characteristics results in very large double values leading to an overflow, which can be avoided using the normalized values.

3.4 Dynamic Selection of Methodology

By means of preliminary experiments with a set of 2000 jobs in CTC (Cornell Theory Center) trace [14], we found that while our filtering techniques improved prediction accuracy in a large number of cases, the original method by Smith gave better predictions in a significant number of cases. This is because while filtering helps reduce noise contributed by dissimilar jobs, it can also remove similar jobs due to large scale filtering. We performed analysis of the target jobs for which the use of unfiltered history gave a better prediction than the use of filtered history, to determine the common characteristics of these jobs. We found that for these jobs, the minimum euclidean distance of the target job from the jobs in its history is greater when filtration is applied.

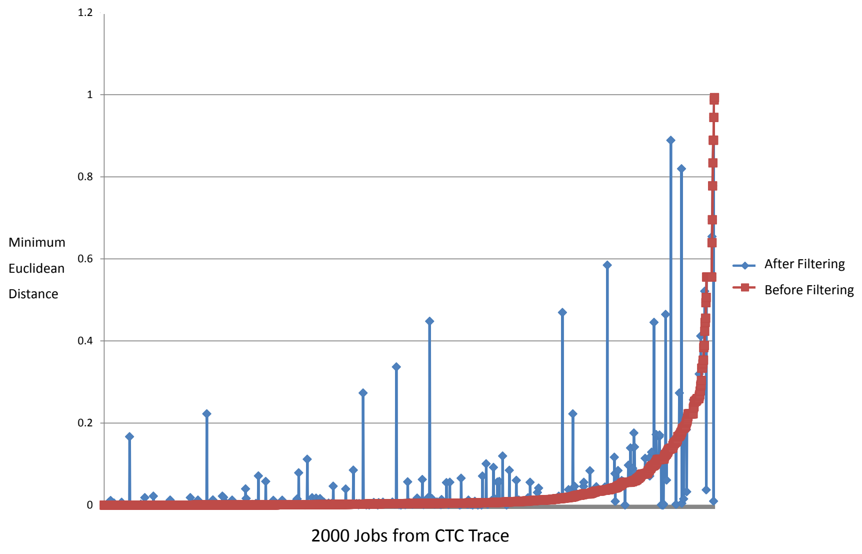

Figure 3.2: Minimum Euclidean Distance Before & After Filtering

Hence we devised a dynamic strategy, in which we obtain predictions from both Smith's technique and filtration technique. We then chose the one with smaller minimum euclidean distance of the target job from its history. This further reduced the mean prediction error.

3.5 Machine Learning Techniques

The task of predicting the run time of a target job based on some of the jobs from history whose actual runtimes are known can be seen as a supervised machine learning problem and the jobs in history as the labeled training data. Regression techniques like support vector machines, decision trees, k-nearest neighbors, ridge regression, lasso regression, least angles regression, bayesian ridge regression and stochastic gradient descent can be used in place of the prediction techniques used by Smith's technique, i.e mean or linear regression. We used the functions provided by the tool, scikit-learn [15]. It is an open source machine learning tool in Python.

We used the history corresponding to the dynamically selected prediction as the training set to various machine learning techniques mentioned above. The inclusion of filtration and machine learning techniques in the existing Smith's technique led to a 14% decrease in the mean prediction error.

Chapter 4

Experiments and results

4.1 Experiment Setup

Table 4.1 shows the job traces used for obtaining the results. The jobs that do not execute completely due to failure or cancellation are removed from all the traces and the remaining jobs are used. The numbers given in Table 4.1 are of the completed jobs. In our technique we used only the first 50000 completed jobs from the traces SDSC Blue and DAS2fs0. The other traces are used completely. The first 10000 jobs in each trace are used as a training set in determining the best template set. Hence, while computing the mean error in prediction of a technique for a trace only the jobs after 10000th job are considered.

Table 4.1: Traces

Trace	Duration (in months)	Number of Completed Jobs	Number Used	User Estimates
CTC SP2	11	62630	62630	Yes
ANL Intrepid	8	68936	68936	Yes
SDSC Blue	32	243314	50000	Yes
DAS2 fs0	12	223068	50000	Yes
SDSC Paragon95	12	53064	53064	Yes
SDSC Paragon96	12	31321	31321	Yes

4.2 Results

4.2.1 Results from Machine Learning Techniques

Table 4.2 shows the mean errors for different traces when different machine learning techniques are used on the history corresponding to the dynamically selected prediction. Most of the techniques showed an improvement over mean/linear regression for most of the traces. Lasso regression gave the best result for CTC. The technique of Ordinary least squares stochastic gradient descent gave the best result for DAS2 fs0. The least errors for SDSC 95 and SDSC 96 are given by Decision trees.

It can be observed that Lasso Regression, Ridge Regression and Decision trees gave better results for all the traces when compared to the use of mean/linear regression on the runtimes of the similar jobs that are chosen after filtering and dynamic selection. Of these 3 techniques, Lasso regression gave the best results. Hence we show comparison results using Lasso Regression in Table 4.3. The mean error values are in seconds.

4.2.2 Results from Smith's Technique, Filtering, Dynamic Selection and Lasso Regression

Table 4.3 compares the results of Smith's technique, our techniques of filtering and dynamically choosing the prediction and for Lasso regression.

Table 4.2: Mean errors for Machine Learning Techniques

Trace	SVM	Decision Tree	K Nearest Neighbors	Ridge	Lasso	Lasso Least Angles	Ordinary Least Squares stochastic gradient descent
CTC SP2	4436.6s	4283s	4466.7s	4125.3s	4049s	-	5015.5s
ANL Intrepid	2810.7s	2840.9s	2826.2s	2811s	2806.6s	2808.8s	3003.5s
SDSC Blue	2539s	2545.4s	2540.3s	2534.9s	2515.9s	2517s	3020.6s
DAS2 fs0	1118.7s	1112.9s	1119.5s	1119.5s	1116.3s	1115.7s	859.93s
SDSC Paragon95	1338.6s	994.46s	1525.7s	1500.5s	1393.5s	1378s	1545.7s
SDSC Paragon96	2288.7s	1439.1s	2375.6s	2318.5s	2007.4s	1966.2s	2413.7s

Table 4.3: Mean errors for different techniques

Trace	Smith's (1)	Filtering + Smith's	Dynamic Method with mean or LR	Dynamic method with Lasso Regression (4)	Percentage Reduction from (1) to (4)
CTC SP2	4748.47s	4528.7s	4291.87s	4049s	14.73
ANL Intrepid	2992.78s	2941.86s	2841.75s	2806.6s	6.22
SDSC Blue	2653.74s	2550.72s	2545.42s	2515.9s	5.19
DAS2 fs0	1268.22s	1146.67s	1124.79s	1116.3s	11.98
SDSC Paragon95	2239.96s	2013.21s	1854.84s	1393.5s	37.79
SDSC Paragon96	3259.81s	3346.17s	2553.92s	2007.4s	38.42

Table 4.4: Percentage Errors

Trace	Smith's	Filtering + Smith's	Dynamic Method with mean or LR	Dynamic method with Lasso Regression	Average Runtime
CTC SP2	48.747	46.491	44.06	41.567	9741s
ANL Intrepid	56.359	55.4	53.515	52.853	5310.2s
SDSC Blue	66.46	63.482	63.35	62.616	4018s
DAS2 fs0	171.91	155.44	152.47	151.32	737.71s
SDSC Paragon95	58.128	52.244	48.134	36.162	3853.5s
SDSC Paragon96	37.643	38.64	29.492	23.181	8659.8s

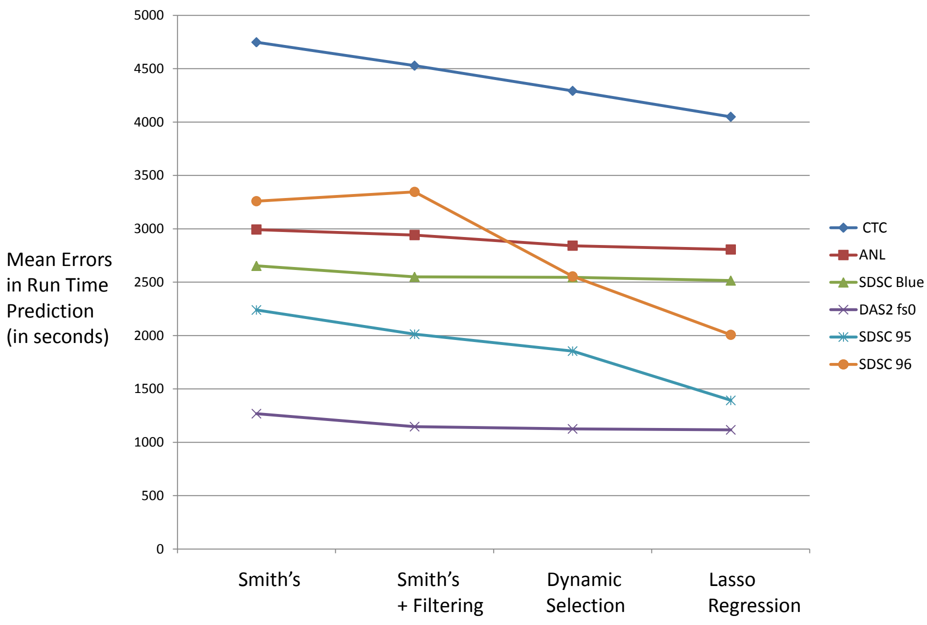

Figure 4.1: The Change in Mean Errors when different techniques are applied

Table 4.3 shows that applying the filtering technique led to a decrease in the mean error, as there was a reduction in the noise in choosing similar jobs in most of the cases. Trace SDSC 96 has an increase of 2% in mean error when filtering is applied. This is because filtering lead to low history for target jobs in the SDSC 96 trace. For traces other than SDSC 96,there is a reduction ranging from 2% to 10% in mean errors when filtering is applied to reduce noise.

Observing the effect of filtering on runtime predictions for 2000 jobs from CTC trace showed that there are significant number of target jobs that have a better prediction with the unfiltered history. These cases showed an improvement when dynamic selection of history is done. The mean errors reduced by 23% for sdsc 96 and by 0.2% - 7% for all the other traces when the dynamic selection is applied. This is because dynamic selection reduces the cases in which similar jobs are eliminated in the process of removing noise. That was the case with SDSC 96, where filtering increased the mean error in runtime prediction. The trace that showed the least improvement among the rest is SDSC 95. This means that the noise in the similar jobs before filtering is applied is very less for these traces, which could be the result of the dedicated nature of the system.

The use of machine learning techniques gave better estimates of runtimes compared to the mean and linear regression techniques that are used in Smith's work. The reduction in mean error when Lasso regression is applied on dynamically selected history is over 20% for SDSC 95 and SDSC 96. It is about 0.7% - 5% for the other 4 traces. This could be because the techniques of mean and linear regression used only runtimes and number of nodes of the similar jobs to obtain prediction, while the machine learning techniques used all the characteristics of the similar jobs for the prediction.

The last column shows the over all improvement obtained by employing our stratagies over Smith's technique. It has the percentage reduction of mean error from Smith's technique to using Dynamic Selection and Lasso Regression. An improvement of 38% is seen for the traces SDSC 95 and SDSC 96, but the traces ANL Intrepid and SDSC Blue have shown only 5% - 6% reduction in mean errors. The traces SDSC 95 and SDSC 96 showed good improvements in all stages except after filtering. The final improvements for these traces are the result of the availability of the similar jobs in the history. For the traces ANL Intrepid and SDSC Blue the jobs chosen by Smith's templates are'nt very similar to the target jobs, this is discussed in the next chapter where the main reason for high error in ANL Intrepid and SDSC Blue is shown to be widely varying runtimes of similar jobs. This could be because of the non-availability of the similar jobs in the past or due to the poor selection by the templates or due to the non-dedicated nature of the system. The changes in the mean errors when each of the techniques is applied is shown in Figure 4.1

Table 4.4 shows the results of runtime prediction in comparison to the actual runtimes. It presents the mean error as a percentage of the mean runtime in the trace. For DAS2 fs0, the prediction errors are very high when compared to the actual runtimes. The mean errors are over 1.5 times the mean runtime. This Other traces have a percentage mean error between 20% and 60%. This is because the average run time for the trace DAS2 fs0 is very small compared to the mean runtimes of other traces. This can be seen in the last column of the Table 4.4.

Chapter 5

Discussion

The properties of the similar jobs from history used for a jobs runtime prediction are observed to analyze the factors resulting in high errors for some jobs. Here, the high error predictions are those with error greater than the mean error.

5.1 Reasons for High Error

The jobs with high error are checked if they have any of the following characteristics.

1. **Large minimum distance:** The minimum of the distances of the target job from the jobs used for its runtime prediction is larger than the mean of the minimum distances.

2. **Poor history:** The number of similar jobs is smaller than the mean of these numbers.

3. **Widely varying runtimes:** The difference between the maximum and minimum of the runtimes in similar jobs is larger than the mean of the runtime ranges.

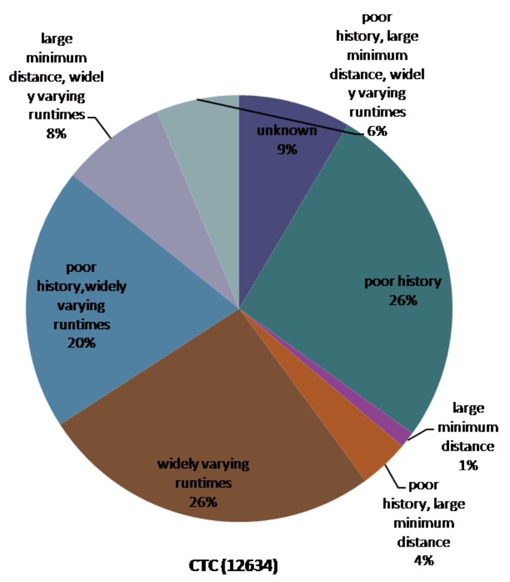

Figure 5.1: CTC trace has 12634 jobs with high error

5.2 Distribution of Jobs According to the Reasons for High Error

Most of the jobs with high prediction error are found to have similar jobs with one or more of these properties. The piecharts in figures 5.1 to 5.6 show the jobs grouped according to the possible reason(s) for the high error in runtime prediction from the traces shown in Table 4.1.

5.2.1 CTC Trace

91% of the high error jobs from CTC had at least one of the 3 properties. Widely varying runtimes and poor history are the main reasons for high error. This is shown in figure 5.1.

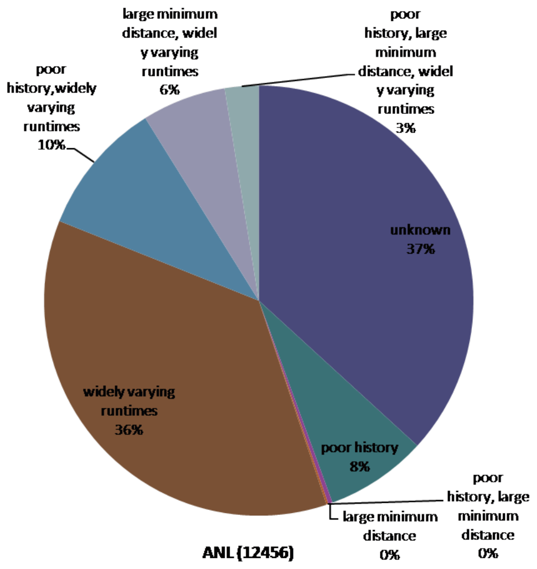

Figure 5.2: ANL trace has 12456 jobs with high error

5.2.2 ANL Trace

63% of the high error predictions from ANL had at least one of the 3 properties. Widely varying runtimes is the main reason for high error, poor history also has a significant role. This is shown in figure 5.2.

5.2.3 SDSC Blue

77% of the high error predictions from SDSC Blue had atleast one of the 3 properties. Widely varying runtimes and poor history are the main reasons for high error. This is shown in figure 5.3.

5.2.4 DAS2 fs0

87% of the high error predictions from DAS2 fs0 had atleast one of the 3 properties. Poor history is the main reason, over 65% of the high error jobs had a poor history. This is shown in figure 5.4.

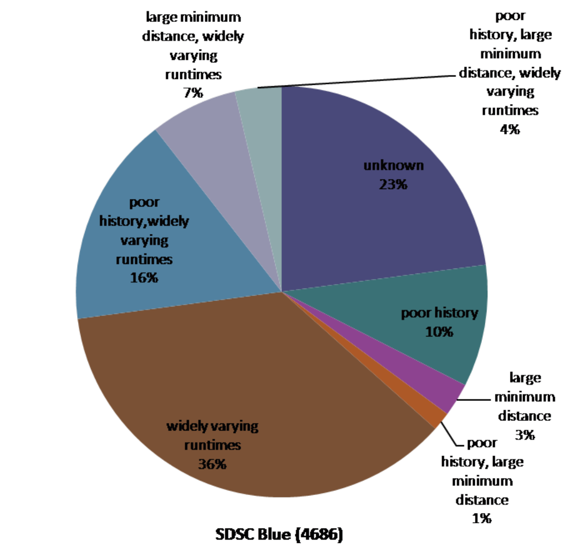

Figure 5.3: SDSC Blue trace has 4686 jobs with high error

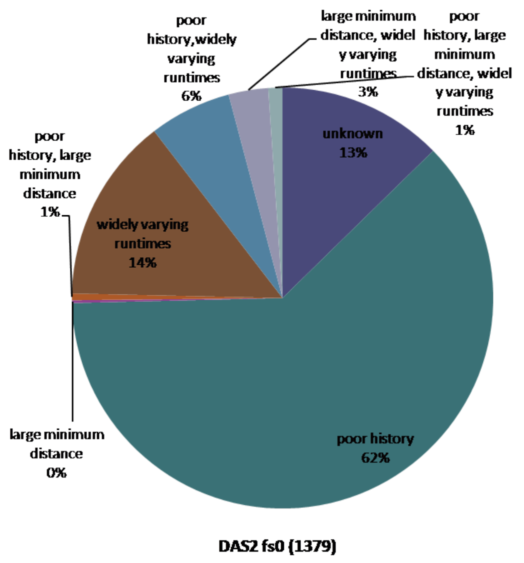

Figure 5.4: DAS2 fs0 trace has 1379 jobs with high error

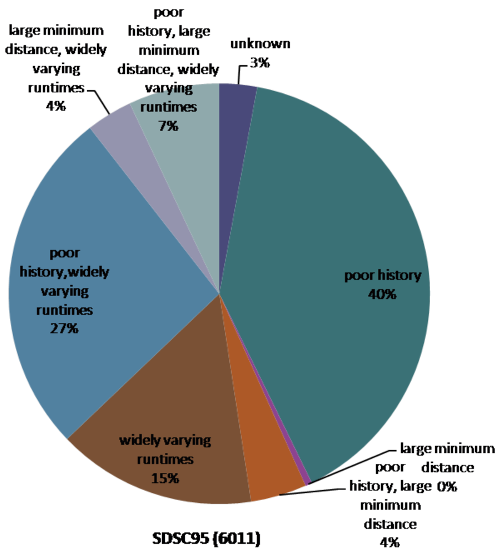

Figure 5.5: SDSC 95 trace has 6011 jobs with high error

5.2.5 SDSC 95

97% of the high error predictions from SDSC 95 had atleast one of the 3 properties and most of the prediction had poor history. This is shown in figure 5.5.

5.2.6 SDSC 96

80% of the high error predictions from SDSC 96 had atleast one of the 3 properties. Poor history is the main reason for high prediction error. This is shown in figure 5.6.

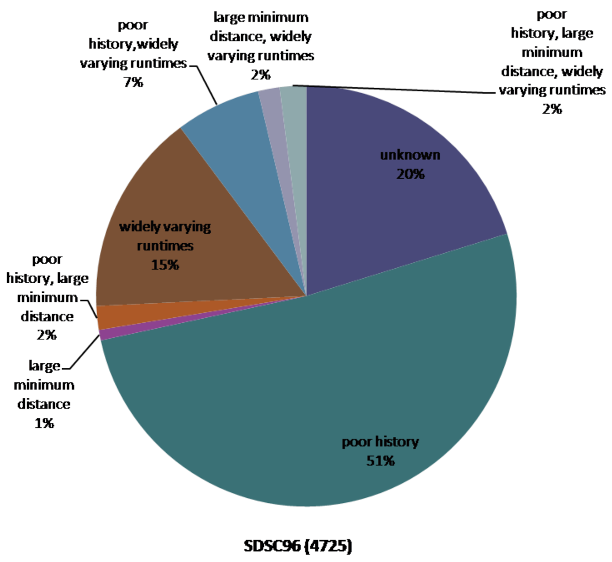

Figure 5.6: SDSC 96 trace has 4725 jobs with high error

5.3 Discussion

Wide Variation in runtimes of similar jobs and Poor history are the main factors contributing to the high errors.

The wide variation in run times can be due to the bad selection of the similar jobs by the templates. It can also arise from the non-dedicated nature of the batch systems, where similar jobs might take very different times for their execution. We tried to reduce the wide variation, that arises from noise in selecting similar jobs, by applying filtering techniques.

Poor history can be the case with jobs that do not have many jobs submitted before it to the batch system that are similar to it. Low history can also result from strict templates or due to too much filtering. We tried to relax filtering and prevent filtering of similar jobs by employing dynamic selection technique.

Large minimum distance is the reason for high errors in less number of cases compared to the other two reasons. We have already chosen the prediction that reduces the minimum distance.

Chapter 6

Conclusions and Future Work

We developed filtering techniques to remove noise in the similar jobs chosen by Smiths technique, devised a dynamic strategy to choose between filtered and unfiltered history and used machine learning techniques for prediction. With a combination of these we got an improvement up to 38% over Smiths technique. We observed that widely varying runtimes is the main reason for high Error predictions in the traces CTC, ANL and SDSC Blue and poor History is the main reason in DAS2 fs0, SDSC95 and SDSC96

Our work can be extended by using the runtime predictions in developing scheduling strategies. More analysis can be done on which machine learning technique works the best in each case, various cases arising from the properties of the history of a target job like the history size, minimum distance of the jobs, runtime ranges etc. Based on this analysis the technique can dynamically choose the machine learning technique for each prediction.

Bibliography

[1] Wei Tang and Narayan Desai and Daniel Buettner and Zhiling Lan, *Analysing and Adjusting User Runtime Estimates to Improve Job Scheduling on the Blue Gene/P*, IEEE International Symposium on Parallel and Distributed Processing, 1–11 (2010).

[2] Dan Tsafrir and Dror G. Feitelson, *The dynamics of backfilling: solving the mystery of why increased inaccuracy may help*, IEEE International Symposium on Workload Characterization, 131–141 (2006).

[3] Ahuva W.Mu'alem and Dror G.Feitelson, *Utilization,Predictability,Workloads and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling*, IEEE Transactions on Parallel and Distributed Systems,(2001)

[4] Yulai Yuan and Guangwen Yang and Yongwei Wu and Weimin Zheng, *PV-EASY A Strict Fairness Guaranteed and Prediction Enabled Scheduler in Parallel Job Scheduling*, HPDC ,240–251 (2010)

[5] H A Sanjay and Sathish Vadhiyar *Performance modeling of parallel applications for grid scheduling*, Journal of Parallel and Distributed Computing, 1135–1145 (2008)

[6] Warren Smith and Ian Foster and Valerie Taylor, *Predicting Application Run Times Using Historical Information*, Journal of Parallel and Distributed Computing, 1007–1016 (2004)

[7] Warren Smith and Valerie Taylor and Ian Foster, *Using runtime predictions to estimate queue wait times and improve scheduler performance*, Job Scheduling Strategies for Parallel Processing, 49 (1999)

[8] Yoav Etsion and Dan Tsafrir, *A short survey of commercial cluster batch schedulers*, School of Computer Science and Engineering, the Hebrew University, (2005)

[9] A.M. Alkindi and D.J Kerbyson and G.R Nudd, *Dynamic Instrumentation and Performance Prediction of Application Execution*, High Performance Computing and Networking, 313–323 (2001)

[10] R.Block and S.Sarukkai and P.Mehra, *Automated Performance Prediction of Message Passing Parallel Programs*, Supercomputing '95:Proceedings of the 1995 ACM/IEEE conference on Supercomputing, 313–323 (1995)

[11] G.Nudd and D.Kerbysin and E.Papaefstathiou and S.Perry and J.Harper and D.Wilcox, *PACE - a Toolset for Performance Prediction of Parallel and Distributed Systems*, The International Journal of High Performane Computing Applications, 228–251 (2000)

[12] Z.Xu and X.Zhang and L.Sun, *Semi-empirical Multiprocessor Performance Predictions*, Journal of Parallel and Distributed Computing, 14–28 (1996)

[13] Andrea Matsunaga and Jose Fortes, *On the Use of Machine Learning to Predict the Time and Resources Consumed by Applications*, International Conference on Cluster,Cloud and Grid Computing, 495–504 (2010).

[14] http://www.cs.huji.ac.il/labs/parallel/workload/logs.html

[15] www.scikit-learn.org