A Machine Learning Approach for Improving Process Scheduling: A Survey

Article *in* International Journal of Emerging Trends & Technology in Computer Science · January 2017

DOI: 10.14445/22312803/IJCTT-V43P101

5 authors, including:

Sreepraneeth Kotaguddam
Georgia Institute of Technology
1 PUBLICATION **1** CITATION

SEE PROFILE

Namratha M.
BMS College of Engineering
15 PUBLICATIONS **20** CITATIONS

SEE PROFILE

A Machine Learning Approach for Improving Process Scheduling: A Survey

Siddharth Dias[1], Sidharth Naik[2], Sreepraneeth K[3], Sumedha Raman[4], Namratha M[5]

1,2,3,4 (IV year B.E., Department of CSE, BMS College of Engineering, Bangalore.)
5(Assistant Professor, Department of CSE, BMS College of Engineering, Bangalore.)

Abstract*—*Improving interactivity and user experience has always been a challenging task. One aspect of this could be to improve process scheduling. This paper is a detailed survey about the attempts that have been made to incorporate machine learning techniques to improve process scheduling. Various approaches to find the appropriate attributes of a process for predicting resource utilization have been discussed here.

Keywords—*Machine learning, Process Scheduling.*

I. INTRODUCTION

The domain of Computer Science which deals with the capability of computers to learn without specifying direct instructions is termed as Machine Learning. The applications of Machine Learning are generally classified as either supervised or unsupervised learning. In supervised learning, the computer is given a data set with corresponding outputs, so that it may learn to predict the output for new instances after training. Unsupervised learning tries to find structure in the data set based on correlation among variables in the data.

Process Scheduling or job scheduling is the activity by which the Operating System (OS) selects an available process from the job queue for execution. This selection is performed by the scheduler. An important element of process scheduling is context-switching, which takes place when the current process is pre-empted. This involves saving of the state of the current process before switching the CPU to another process. For each context switch, there is an associated overhead which results in loss of valuable processor time slices.

One way of improving a user's experience is to ensure that the processes they use are given a larger share of the resources i.e. more priority. These processes need to be identified and their performance can then be improved using data from previous executions.

II. LITERATURE SURVEY

Before diving into machine learning techniques, characteristics of a process, which contribute significantly to the prediction of the amount of required resources, need to be determined. This has been attempted by analysing data regarding previous executions [1]. The processes were classified into "interactive" and "no interactive" programs. Benchmarked Linux programs were chosen for experimentation. A total of 24 attributes were selected to be used to predict the total execution time of the programs. Required system calls were made to ascertain the values for the various chosen attributes of a particular process.

The first part of this work was the data collection phase where the programs were run for varying input sizes. The collected data was then put into 20 classes to be used for machine learning in WEKA using the "Trees, Lazy, Rules" classifier types supplied. Decision Trees, K-NN and Decision Tables were used for finding robust and accurate predictions. The authors did not limit themselves to just labels but went a step further by analysing real-time characteristics of a process. Various subsets of the attributes were tested in order to determine which subset produced the best predictions for "total execution time". An "attribute evaluator", which assigned a weight to each subset, and a "search method", which determined the kind of search to be performed, were used in the process. Search methods used were: Genetic Search, Best First Search and Rank Search. Evaluation methods employed were: CfsSubsetEval and Consistency SubsetEval.

The best attribute they found was "input size" and "page reclaims" turned out to be the next best. Other attributes that came close were .text (executable instructions), .data (initialized data), .rel.plt, .dynamic (dynamic linking information), page faults and .plt (procedure linkage table). A good prediction rate of (91.4%-99.7%) was achieved using the aforementioned techniques. These results were used to improve PBS scheduling where the resource requirements were estimated based on knowledge base developed by keeping track of previous executions of the program. Thus the authors concluded that using suitable attributes, a good estimation about the process execution behaviour of known programs can be achieved.

In [2], a similar attempt was made by applying data mining techniques to the data present in the kernel about each process, to automatically detect and

group the processes which have similar behaviour and to classify a new process accordingly. They analysed the data to find patterns (with or without intervention) using classification rules, regression, clustering, sequence modelling and association rules.

The three major groups into which the processes are usually divided are batch, daemon and interactive. The attributes of the processes present in the Linux context were extracted and used to generate a training base. The attributes were grouped using an unsupervised learning algorithm. The groups formed were manually analysed. Each process was identified by the group and a base was generated with the labelled data. The dimensionality of the attribute vector was reduced through algorithms to generate new training bases. The classification algorithms were then used to perform training samples on the reduced bases. The performance of each of the classification algorithm was evaluated in relation to the hit rate and the processing time to achieve the proposed objective. No prior knowledge about the processes was present. The data was provided in two phases - learning and testing. The source of data was a remote terminal server used by undergraduate students at PUCPR for academic purposes. The attributes were extracted from /proc directory in the server as server/proc using PERL. The data was later fed into WEKA for processing. This data was collected twice a day for 185 days to form a database of 97,391 distinct samples each having 108 attributes.

The grouping analysis used unsupervised learning algorithms to group the data based on various parameters. The results of the analysis classified processes into 6 different groups:

- A (Interactive applications): all types of interactive processes (editors of text, Web browsers, email clients, etc.).
- D (Daemons): processes that run in the background and are ready to instructions.
- F (Desktop Features): processes that perform tasks to support the graphical desktop environment (configuration, component framework, etc.).
- N (Network): processes involved with network communication.
- C (Text Commands): simple text-mode terminal commands.
- K (Kernel threads): inner threads of the core of the operating system.
- O (others): processes that do not fit into the other groups.

The next step was aimed at finding the most important attributes which could fully describe all the important features with minimal redundancies. The algorithms used to find the best subsets were Genetic Search, InfoGain: Ranker, CFS: Rank Search and CFS: Best-First Search. The evaluation procedures were chosen as Information Gain and CFS methods. Through the classification algorithms, they were able to obtain four databases and then a database was generated with the common attributes.

The complete database consisted of 108 attributes. The database formed by Genetic Search, CFS: Rank Search, InfoGain: Ranker and CFS: Best-First Search returned 40, 19, 10 and 9 attributes respectively. Only 4 common attributes could be found - stat: vsize (virtual memory in use by the process), statm: text (size of memory used by process code), stat: endcode (address below which the process code can execute), status: VmLib (space used by process shared libraries).

To categorize the new processes into the predefined classes based on common characteristics, four methods were chosen, namely C4.5, OneR, Naïve Bayes, MLP (Multi-Layer neural networks). The best results were obtained from C4.5 and OneR. With just 4 attributes, the results obtained had a hit rate greater than 95%.

In [3] the authors proposed a new scheduling methodology called "semantical cognitive scheduling". The proposed scheduler aimed at providing a "cognitive" approach to scheduling. The usefulness of a process in a system was estimated with the help of a "utility value". The approach was based on the fact that even though there are a diverse range of processes in the system, the types of applications, if grouped appropriately, are relatively small. The author used several approaches to determine the utility value such as: "White box" (software vendor specified), "Black box" (utilization of AI methods to classify applications), "Offline Learning" (classify applications separately and use results during execution) and "Online Learning" (learning from user feedback).

Based on the lemmas proved by the authors, they presented two algorithms for determining the utility value. The Optimal Algorithm used Dynamic Programming Approach to reduce computation complexity when the number of possible states of the system was reasonable. The second algorithm used Greedy Technique to optimise performance if the number of possible states was small. These results were applied to improve the management of services and daemons which tend to increase in number over time affecting boot time and performance.

An approach which utilises dependencies among services and user applications was proposed along

with the possibility of loading services in such a manner that the highest utility of the system was achieved. Statistical analysis of previous executions of services can be used to estimate best combination of services. These steps can improve the interactive experience of the user.

In [4], the authors attempted to utilize machine learning techniques to predict the CPU burst-time for different processes. Scheduling algorithms, such as Shortest Job First (SJF) and Shortest Time Remaining First (SRTF), rely on predictions of the CPU burst lengths of processes in the ready queue. Methods such as Exponential averaging have been employed previously in the prediction of CPU burst times, but their results have not always been reliable or accurate.

The proposed approach was to identify attributes of processes that could help determine their CPU burst lengths. These were then used by machine learning techniques for prediction. This was done in the following manner:

- Data sets of real workloads containing process attributes were utilized. These were divided into training and testing sets.
- Relevant attributes were selected and divided into two categories based on whether historical data was available or not.
- Different ML algorithms, namely SMOreg, MLP, Decision tree and K-NN, were trained on both the categories of training data sets and models were generated and evaluated using the testing datasets.
- Two criteria for evaluation were used, namely correlation coefficient (CC) and relative absolute error (RAR). These were used to determine how close the predicted values of CPU bursts were to the actual values. For a good result, a high value of CC accompanied by a low RAR percentage was required.
- Relief selection technique, implemented on the WEKA tool, was then used to rank the attributes according to their significance in determining the CPU burst lengths.
- From all of the above, tables were constructed for both variations (with and without history) of the datasets and analysed to determine the best set of attributes as well as the best ML technique.

It has been found that using ML techniques in tandem with attributes of processes with historical information gave tremendous results in accurately predicting the CPU bursts of different processes, as compared to using attributes without historical data. All of the aforementioned ML techniques gave a high CC value accompanied by a low RAR. Out of these, k-NN was found to be the most efficient algorithm, giving a CC value of 0.9933 along with the RAR being as low as 3.46%. Thus, employing machine learning techniques in the prediction of CPU burst lengths is highly beneficial and much more accurate than current techniques. Also, the attribute selection techniques improve the performance in terms of space complexity.

In [5] the authors presented a feasible approach to incorporate current machine learning techniques to improve process scheduling by allocating variable time slices for different processes to reduce the overhead of context-switching. Processes were associated with a single integer field referred to as special_time_slice (STS), which helped in indicating the best estimate ofCPU cycles to be allocated, so as to minimize their turnaround time. The processes were categorised into different STS classes, each having an interval of 50 ticks.

Determining the STS class of a program necessitated prior mapping. However, since an infinite number of process instances exist in the real world, keeping track of each and every instance is a daunting task. To reduce this complexity, machine learning tools were employed as follows:

- A set of 5 standard programs were chosen. Different instances of these were run with different values of STS to determine the shortest turnaround time for each. The data of each instance, including process attributes, was aggregated into one data set.
- The best STS for a program instance determined its STS class.
- Different machine learning techniques such as C.45, k-NN etc. were trained with this data set (learning). They were then tested to determine the accuracy with which they could predict the best STS for each process.
- Operating on as many as 18 attributes is an uphill task for the ML techniques. Hence, Exhaustive and Genetic search methods were used to determine the minimum set of attributes that could accurately determine the STS class of any process.
- When a new process arrived, the above mentioned process was used to classify and obtain its STS value. This was then fed to the Kernel.

Hence, a mapping was established between process attributes and STS classes. C.45 was found to be the best classifier and 6 characteristics were identified to help in prediction. It has been shown that with the usage of special_time_sllices, there was a noticeable drop in the turnaround time of various processes. Initial experiments showed a drop of around 1.4% - 5.8%, and with bigger data sets over time, the numbers were predicted to be even higher.

However, a major drawback of this method is that whenever a new program arrives, there is an overhead of classifying it separately and feeding that data to the Kernel, deteriorating user experience. Also, having to maintain the entire dataset on the system would lead to space constraints, as the amount of data was bound to increase over time. The authors have mentioned security flaws in the system where a user can modify data sets, leading to haphazard results. However, the work is still a major stepping stone in the field of modern age Operating Systems.

In [6], the history of a Linux scheduler was used to predict parameters of current processes, namely turnaround time (TaT) and number of context switches. This information was then used to determine an improved time quantum value, consequently reducing the number of pre-emptions performed by the scheduler. By lowering this number, the CPU overhead owing to pre-emption and context-switching was considerably minimized. The adaptive time quantum value was estimated based on a refined set of attributes that were found to be pivotal in building the classifier, including but not restricted to the size of uninitialized memory, size of hash table, size of dynamic linking table and size of string table. These attributes were used to create a Decision Tree that classified instances into 20 categories of ranges of time slices. The classifier was modelled using WEKA. A self-learning module was implemented based on Reinforcement Learning (RL). The decision obtained from the classifier was given as input to this module. While the Decision Tree classified a new process, this module ran in the background to examine new classes. The decision of the scheduler to allocate time quanta was modelled as a Markov Chain Process. Hence, previous classifications were not taken into consideration. The effectiveness of the model was analysed by executing a set of programs multiple times with different priorities. This could be evaluated by calculating the number of CPU cycles that were saved as the time quantum varied. To understand how the TaT is affected by the time slice class assigned, multiple graphs were plotted for common programs such as matrix multiplication, merge sort and heapsort. The correlation was found to be nonlinear; this emphasized the need to design a suitable classifier that can predict the time quantum optimally.
This work supports the proposition that minimizing the number of context switches can significantly reduce the TaT by assigning relevant time slice classes to each process. The self-learning module provided the system with the ability to constantly improve the classification process.

III. Conclusion

In conclusion, it is seen that machine learning techniques can be efficiently integrated into existing operating systems to deliver a seamless user experience. Initial experiments have been fairly successful in pinpointing specific attributes of processes that are better suited than others in predicting CPU burst cycles and resource utilisation. Classification of processes has been extensively employed in previous works to simplify decision making. Granting extra time slices based on analysis of previous executions of a process was an innovative approach to improve performance. However, the scheduler did not vary the time slice of a process based on the frequency of usage of the application. Future work in this domain can entail prioritising programs that are used routinely. There is scope for improving scheduling to cater better to the needs of the user.

ACKNOWLEDGMENT

The work reported in this paper is supported by the college [BMSCE, Bangalore] through the TECHNICAL EDUCATION QUALITY IMPROVEMENT PROGRAMME [TEQIP-II] of the MHRD, Government of India.

REFERENCES

[1] A. Negi and K.P. Kishore. “Characterizing Process Execution Behaviour Using Machine Learning Techniques”, In DpROMWorkShop Proceedings, HiPC 2004 International Conference, December, 2004.

[2] Araujo, Priscila Vriesman, Carlos Alberto Maziero, and Júlio César Nievola. “ClassificaçãoAutomática de ProcessosemSistemasOperacionais. Diss. Dissertação de mestrado”, 74 p. Pós-GraduaçãoemInformática, PontifíciaUniversidadeCatólica do Paraná, Curitiba, 2011.

[3] ShlomiDolev , Avi Mendelson and IgalShilman,” Semantical cognitive scheduling”, Nov. 2012 Technical Report # 13-02 Research in part by Microsoft and (Fronts).

[4] Helmy, Tarek, Sadam Al-Azani, and Omar Bin-Obaidellah. "A Machine Learning-Based Approach to Estimate the CPU-Burst Time for Processes in the Computational Grids”, In 2015 Third International Conference on Artificial Intelligence, Modelling and Simulation, 2015.

[5] A. Negi and K.P. Kishore. “Applying machine learning techniques to improve linux process scheduling”, In TENCON 2005 IEEE, Region 10, pages 16, Nov. 2005.

[6] Ojha, Prakhar, et al. "Learning Scheduler Parameters for Adaptive pre-emption”, CS & IT-CSCP 2015.

[7] Silberschatz, Abraham, Peter B. Galvin, and Greg Gagne. Operating system concepts. Vol. 8. Wiley, 2013.

[8] Tom Mitchell, Machine Learning, 1st edition, The Mc-Graw Hill Company Inc. International Edition,1997