|  |  |
| --- | --- |
|  | Process Scheduling using Machine Learning |
|  |  |
|  | Daniel Houwen  Sai Kishore  COEN283 – Operating Systems  SPRING 2014 |

Table of Contents

[2 Introduction 2](#_Toc387776994)

[2.1 Objective 2](#_Toc387776995)

[2.2 Problem background 2](#_Toc387776996)

[2.3 Application of the problem to operating systems 2](#_Toc387776997)

[2.4 Disadvantages of other approaches 3](#_Toc387776998)

[2.5 Advantages of the machine learning approach 3](#_Toc387776999)

[2.6 Problem statement 3](#_Toc387777000)

[2.7 Project scope 3](#_Toc387777001)

[3 Theoretical Basis and Related Literature 3](#_Toc387777002)

[3.1 Problem definition 3](#_Toc387777003)

[3.2 Theoretical background 3](#_Toc387777004)

[3.3 Related research 3](#_Toc387777005)

[3.3.1 Research 1 - Advantages and Disadvantages 3](#_Toc387777006)

[3.3.2 Research 2 – Advantages and Disadvantages 4](#_Toc387777007)

[3.4 Our solution 4](#_Toc387777008)

[3.4.1 Differences from Existing Solutions 4](#_Toc387777009)

[3.4.2 Advantages over Existing Solutions 4](#_Toc387777010)

[4 Hypothesis 4](#_Toc387777011)

[5 Methodology 4](#_Toc387777012)

[5.1 Data Collection 4](#_Toc387777013)

[5.2 Proposed Solution 4](#_Toc387777014)

[5.2.1 Algorithm Design 4](#_Toc387777015)

[5.2.2 Programming Environment 4](#_Toc387777016)

[5.2.3 Other Tools Used 5](#_Toc387777017)

[5.2.4 Prototype? 5](#_Toc387777018)

[5.3 Output Generation 5](#_Toc387777019)

[5.4 Testing Methodology 5](#_Toc387777020)

[6 Implementation 5](#_Toc387777021)

[6.1 Code Design 5](#_Toc387777022)

[7 Data Analysis 5](#_Toc387777023)

[7.1 Output 5](#_Toc387777024)

[7.1.1 Output Analysis 5](#_Toc387777025)

[7.1.2 Hypothesis Comparison 6](#_Toc387777026)

[7.1.3 Abnormal Cases 6](#_Toc387777027)

[7.2 Statistic Regression 6](#_Toc387777028)

[7.3 Discussion 6](#_Toc387777029)

[8 Conclusion 6](#_Toc387777030)

[8.1 Summary 6](#_Toc387777031)

[8.2 Future Studies 6](#_Toc387777032)

[9 Bibliography 6](#_Toc387777033)

[10 Appendices 7](#_Toc387777034)

[10.1 Appendix A – Program Flowchart 7](#_Toc387777035)

[10.2 Appendix B – Source Code 7](#_Toc387777036)

[10.3 Appendix C – I/O Listing 7](#_Toc387777037)

# Introduction

## Objective

The aim of this project is to develop a task scheduling method using supervised machine learning techniques that is more efficient than common existing methods. Since there many different techniques, with unique advantages and disadvantages, finding a suitable technique will be an important part to develop efficient scheduling.

## Problem background

In many areas of computing, developing efficient task scheduling algorithms can significantly increase the performance of the system. This is particularly true with parallel and distributed computing, which is becoming increasingly widespread. There has been much work in developing heuristic methods for task scheduling and using machine learning techniques for optimization problems, but there is still a lack of research combining the two areas.

## Application of the problem to operating systems

Process scheduling is one of the core concepts in operating system theory, because it is one of the most important tasks for the efficient operation of modern operating systems. This project is investigating an innovative method for process scheduling, which can lead to the improved performance of a system.

## Disadvantages of other approaches

There have been many developments in process scheduling leading to better performance, however as systems are becoming more sophisticated, the approaches needed to optimize these systems must also become more sophisticated. There are several new approaches to task scheduling and mapping being applied to same systems as this approach. These include nondeterministic approaches, such as genetic algorithms or unsupervised learning algorithms, and deterministic approaches, such as directed acyclic graph scheduling algorithms. Nondeterministic approaches may provide near-optimal scheduling, but they increase the execution time of the system which reduces their efficiency. Deterministic approaches have the advantage of efficient computation, but their actual performance is typically worse than non-deterministic approaches.

## Advantages of supervised learning

Developing a supervised learning algorithm is a deterministic approach, which means it has the advantage of fast execution. To overcome the performance issues associated with deterministic approaches, this approach is concerned with developing an accurate classification of task processing time, so that the best scheduling decisions may be made for each task. Where other deterministic approaches have little to no information about how tasks will execute, a supervised learning approach can attempt to learn how tasks will execute before the system even starts.

## Problem statement

Given the increasing complexity of processor architectures, sophisticated approaches are needed for efficient scheduling and mapping on these systems. Incorporating machine learning techniques has provided improved results in many related fields, so it is worthwhile to apply these techniques to scheduling problems.

## Project scope

The first portion of this project will be determining which type of machine learning algorithm to use, which is tied to the specific scheduling aspect to be optimized, so it will be investigated in parallel. The implementation will cover feature extraction, developing training data, applying the training data, and implementing the algorithm in a scheduling program, using different metrics of testing against a common existing scheduling method.

# Theoretical Basis and Related Literature

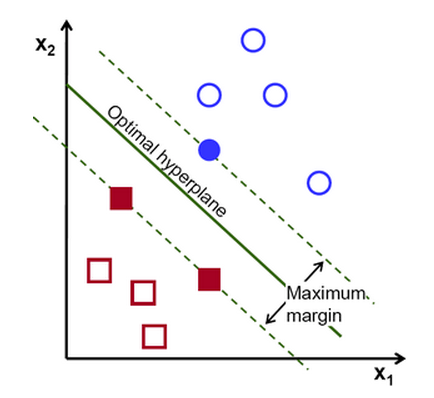
## Problem definition

The reason efficient scheduling algorithms are needed is because of the requirement of most operating systems to perform multitasking. There are many requirements for a scheduling algorithm, including but not limited to: throughput, turnaround time, fairness, waiting, and optimal CPU utilization. As computing systems increase in complexity and become more diverse, more robust algorithms are needed to handle the new requirements of these systems.

## Theoretical background

Process scheduling is not a new problem in operating systems. Many efficient algorithms have existed for quite some time, and new algorithms are continually being developed to address the scheduling issues involved with emerging computing systems.

The basis of machine learning is creating representations of data, or features of a system, and functions that evaluate the features to determine the output of the system. Supervised learning is an implementation of machine learning where the function, or classification, is created using known outputs for given test inputs. Support vector machines are models of supervised learning that are defined by a hyperplane separating categories of data, with the maximum margin between categories. A simple example of classification using an SVM in 2D space can be seen in Figure 1.



**Figure 1 – Support Vector Machine Classification in 2D. [5]**

## Related research

“Intelligent Task Mapping using Machine Learning,” by Tetzlaff and Glesner, discussed a method to use machine learning to determine runtime behavior for processes in heterogeneous multiprocessor systems by analyzing static code features [1]. “Applying Machine Learning Techniques to Improve Linux Process Scheduling,” by Negi and Kumar used ML techniques to learn CPU time-slice utilization, and minimize task turnaround time in Linux [2]. “Task Scheduling for Heterogeneous Computing based on Learning Classiﬁer System,” by Yang, Xu, and Jia, used the XCS algorithm to determine the optimal task assignment and execution sequence for heterogeneous systems [3].

### Tetzlaff and Glesner - Advantages and Disadvantages

Although they introduced methodology for a complete approach to ML task scheduling, Tetzlaff and Glesner implemented only supervised learning of loop bounds. Using an unspecified algorithm, they were able to correctly classify the boundaries of loops within tasks within a mean absolute error of 0.94 for 115 static code features. Given their planned implementation for complete scheduling, these results are promising for developing efficient task mapping to reduce interprocessor communication. However, the results are limited, and there is no determination whether or not this approach actually does result in reduced turnaround time or throughput.

### Negi and Kumar – Advantages and Disadvantages

Sai – add this section.

### Yang, Xu, and Jia – Advantages and Disadvantages

Yang, et al., used the XCS algorithm, which combines reinforcement learning with genetic algorithms, to develop efficient task mapping to heterogeneous systems, similar to Tetzlaff and Glesner. In this paper, they were able to develop a complete implementation, finding the optimal task assignment for each processor as well as the task execution sequence. They compared their implementation to the well-known HEFT (Heterogeneous Earliest Finish Time) algorithm, and found that their approach provided a speedup in all test cases. The limitations of this paper are that they only provided a comparison to one other scheduling algorithm, and that their approach used on-line methods, which requires additional computation at runtime.

## Our solution

In this project, support vector machines will be implemented using the SVMRank program in the Minix 3 operating system. SVMRank is an SVM instance specifically designed for efficient training of SVMs to rank data [6]. Minix 3 is a small and simple Unix-like system, well-suited for experimentation and modification [7]. An algorithm will be generated by testing many different example programs set with different priority levels to determine which order of tasks provides the most efficient scheduling. By applying the SVM algorithm to rank the tasks, the default scheduler in Minix, a multi-priority round robin scheduler, will be modified to choose the next process based on the results of the algorithm. The results of this implementation will be compared to the results provided by the default scheduler.

### Differences from Existing Solutions

The main difference from existing solutions is the implementation of an SVM to create efficient scheduling. While SVMs have been applied in some scheduling applications [8] [9], indicating the possibility for use in process scheduling, there is little information on their implementation in this area.

### Advantages over Existing Solutions

The main advantage over deterministic approaches is that this implementation should have little effect on runtime speed, as the training is beforehand. In this case, a ranking SVM is used in this solution which should provide higher accuracy over other non-deterministic ML techniques, because it is specifically tailored for efficient ranking. This combination of accuracy and efficiency should provide advantages over existing solutions.

# Hypothesis

Using an algorithm generated by the RankSVM program, with training data provided by custom tests, more efficient process scheduling will be achieved than the existing round robin scheduler in Minix 3.

# Methodology

## Data Collection

Training data is collected by running many different programs and logging the results

## Proposed Solution

### Algorithm Design

Text.

### Programming Environment

This project is coded using C/C++ on the Koding development environment. Koding is a cloud-based collaborative programming environment, allowing users to code and run programs on Ubuntu VMs [10].

### Other Tools Used

Minix 3 is used as the testing environment, and its default round robin scheduler is modified to incorporate the SVM algorithm. SVMRank is used to train and implement the aforementioned SVM algorithm.

## Output Generation

Text.

## Testing Methodology

Text.

# Implementation

Text.

## Code Design

Text.

# Data Analysis

Text.

## Output

Text.

### Output Analysis

Text.

### Hypothesis Comparison

Text.

### Abnormal Cases

Text.

## Statistic Regression

Text.

## Discussion

Text.

# Conclusion

Text.

## Summary

Text.

## Future Studies

Text.

# Bibliography

[1] Tetzlaff, D.; Glesner, S., "Intelligent Task Mapping Using Machine Learning," *Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on* , vol., no., pp.1,4, 10-12 Dec. 2010

[2] Negi, A.; Kishore, K.P., "Applying Machine Learning Techniques to Improve Linux Process Scheduling," *TENCON 2005 2005 IEEE Region 10* , vol., no., pp.1,6, 21-24 Nov. 2005

[3] Jiadong Yang; Hua Xu; Peifa Jia, "Task Scheduling for Heterogeneous Computing Based on Learning Classifier System," *Artificial Intelligence and Computational Intelligence, 2009. AICI '09. International Conference on* , vol.3, no., pp.370,374, 7-8 Nov. 2009

[4] Andrew S. Tanenbaum. 2007. *Modern Operating Systems* (3rd ed.). Prentice Hall Press, Upper Saddle River, NJ, USA.

[5] “Introduction to Support Vector Machines,” OpenCV 2.4.9.0 Documentation, [online] 2014, <http://docs.opencv.org/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html>

[6] Thorsten Joachims, “SVMrank: Support Vector Machine for Ranking,” Cornell University, [online] 2009, <http://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html>

[7] “More about MINIX 3,” MINIX 3, [online] 2014, <http://www.minix3.org/other/read-more.html>

[8] Yi-Hung Liu; Han-Pang Huang; Yu-Sheng Lin, "Dynamic scheduling of flexible manufacturing system using support vector machines," *Automation Science and Engineering, 2005. IEEE International Conference on* , vol., no., pp.387,392, 1-2 Aug. 2005

[9] Ci Chen; Binghai Zhou; Lifeng Xi, "A support vector machine based scheduling approach for a material handling system," *Natural Computation (ICNC), 2010 Sixth International Conference on* , vol.7, no., pp.3768,3772, 10-12 Aug. 2010

[10] “What is Koding?” Koding Knowledge Base, [online] 2014, <http://learn.koding.com/what-is-koding/>

# Appendices

## Appendix A – Program Flowchart

Text.

## Appendix B – Source Code

Text.

## Appendix C – I/O Listing

Text.