MACHINE LEARNING-BASED LINUX PROCESS SCHEDULER

Students: Nate, removed

Supervisor: removed

Semester: Winter 2020

Lab: Control Robotics and Machine Learning Laboratory

Table of Contents

Abstract

This project seeks to offer a new solution to Operating System Scheduling, an area of research that's been around several decades, this project attempts to bring the technology into the modern age using Machine Learning.

The Project outlines the problems facing current OS schedulers, and why the necessity for a modern approach is a must.

We will go through the literature out there on the subject, what solutions have already been offered. The different advantages and disadvantages each one brings, and how our research differs from what is already out there.

We will go through the process we under went trying to discover a solution, the different evolutions of our solution, what we learned along the way.

We will publish our results and lay out the conclusions we've drawn from them.

Finally we will end with how this project can be used as a stepping stone towards other objectives, some of which we have marked to ourselves as interesting areas we want to expand in further projects.

Motivation

Increase in Computational Power

In recent history we have witnessed an expansion of the human knowledge unparalleled in history, enormous break throughs in the science, medical research, transportation, economic and most notable in the computation and digital world. Like all revolutions the eventual outcomes are still uncertain, though general consensus is that this expansion will accelerate before it slows down.

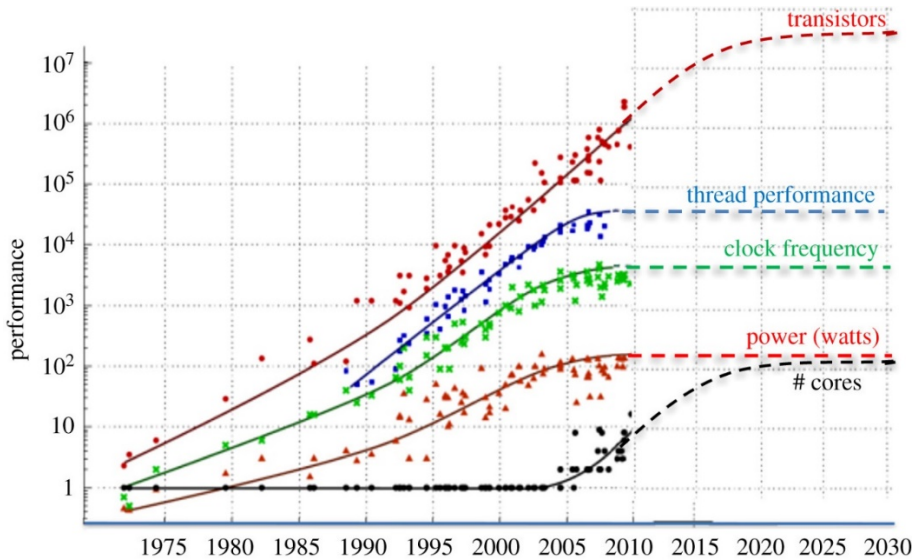
The famous Moore's law predicted that the number of transistors on a microchip doubles every two years, it is famous because over the past fifty years it turned out to be accurate. A definite example of the progress we've made is the sheer fact that more than half the people in the world walk around with a device in their pocket 1,300 times more powerful than the computers used to guide a mission to the moon fifty years ago.

This leap in computation has brought many benefits, but has come with its own set of obstacles. Increase in power consumption, increase in need for heat dispersion, larger batteries, etc.

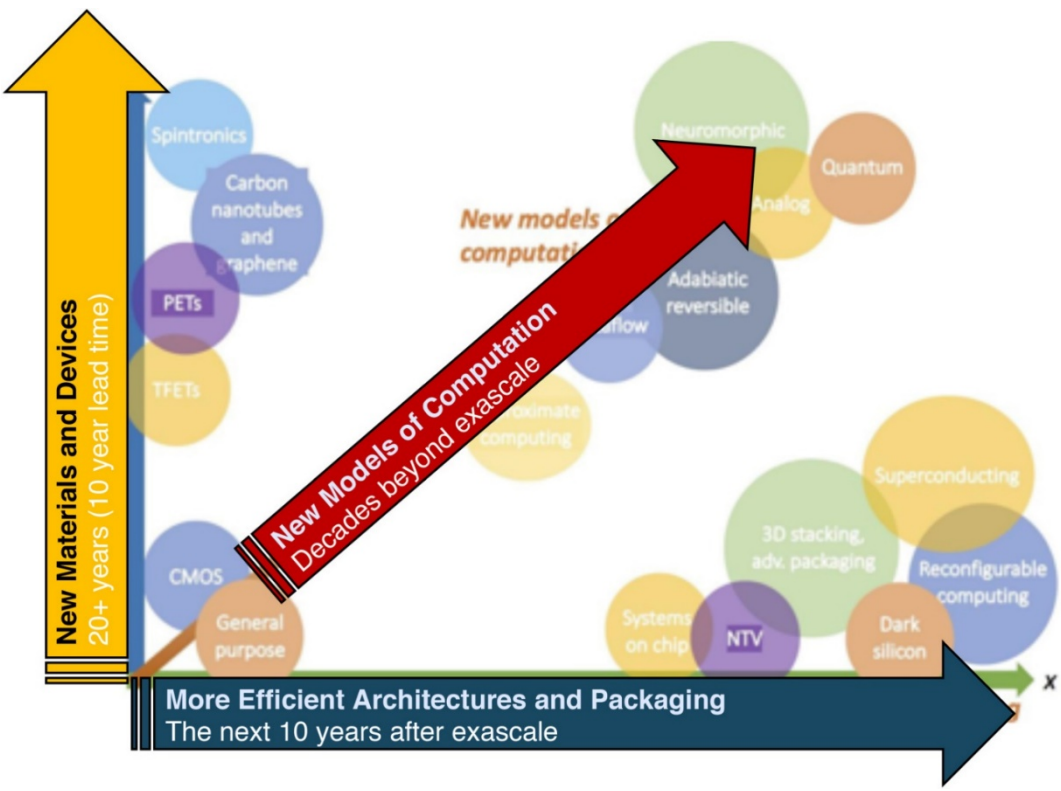
Throughout our degree we studied multiple technologies, some seemingly archaic, some very modern. Noticing this discrepancy we began to question, could this uneven playing field cause problems as the gap grows?

It seemed we weren't the only ones questioning this. Throughout our literary survey for this project we came across an article[1] by John Shalf who went a step further and claimed that not only was there a gap within the technology world, but we can already feel the effects holding us back.

The author claimed that we can no longer count on Moore's law and expect similar increases in computation in the near future. He goes into detail about how he predicts that we have reached a limit and we are now required to seek advancement in computation power through other means.



He claims that we have three avenues which we may pursue. More Efficient Architectures and Packaging, New Materials and Devices, or New Models of Computation. Each avenue with its corresponding time benefits.



This projects fits in with "New Models of Computation". If we can answer our research question and find a new improved way to handle a systems resources we will have helped make one small step in what may be the next advancement of computational technology.

What is a Scheduler

Many very advanced systems are still running legacy software, there exists many cases where the hardware is revolutionized yet the software does not adapt. One such case is the Operating System Scheduler.

The OS Scheduler handles all tasks (we will uses process interchangeably with task) requesting run time on a system. It is a central piece of any system with regards to computation time.

The OS Scheduler have several responsibilities:

1. Foremost the scheduler attempts to maximize hardware resources usage.
2. The scheduler is responsible for allocating resources based on priority.
3. The scheduler is responsible in prevented system “locking”, i.e. any task acquiring a resource and not relinquishing the resource.
4. Lastly it is also designed to give the user an appearance of “multi-tasking” this is done by iterating through different tasks at high frequency, progressing each task incrementally.

Many Scheduler algorithms are very old, for the classic “Round Robin” scheduling algorithm we found a citation as early as 1986[2].

There are several reasons[3] why although the hardware has progressed greatly in 35 years, the scheduler is lagging behind. The two main reasons are money and reliance. As the scheduler algorithms are common across multiple operating systems, changes to them come at great cost. The second reason, reliance, scheduling algorithms are very delicate as they sit at the core of any system, any mistake made by the scheduler has the potential to bring down the system. Fearing this most users prefer to use algorithms that have been proven time and time again to be reliant and stable.

Even though given these two constraints, we believe that we have reached a point where relying on legacy software is no longer a viable option. As the scheduler is now becoming a bottle neck for future systems. Exploring a replacement using Machine Learning seems to have promising potential.

This idea although not unique and has been brought forth before, we will expand on this in the “Literature Survey” chapter. Still requires much research to reach a definitive answer.

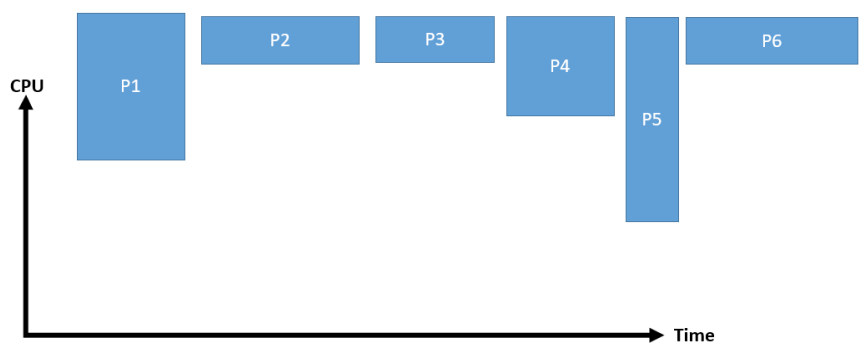
Scheduling Criteria

There are many ways to measure the performance of a scheduler:

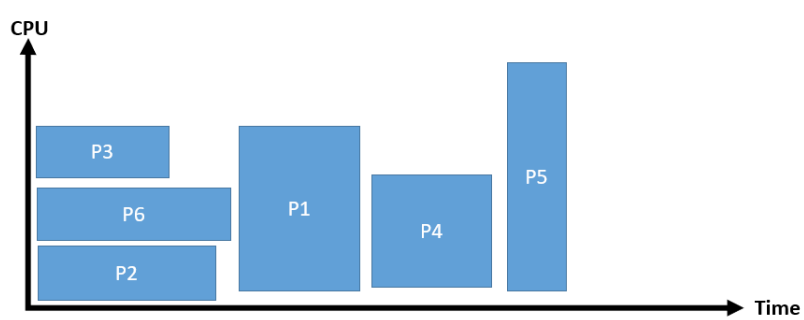
1. **Resource utilization** – How much of the a given resource's time is used to progress a task out of overall run time (ranges between 0-100%).
2. **Throughput** – Number of processes that complete their execution in a given time unit
3. **Turnaround Time** – Amount of time to execute a given process
4. **Waiting Time** – Amount of time a process has been waiting in the "ready queue"
5. **Response Time** – Amount of time it takes from when a request to run has been submitted until a first response is produced.

We will focus on Resource utilization (specifically on **CPU utilization**) as the metric for which we will gauge different algorithms against each other.

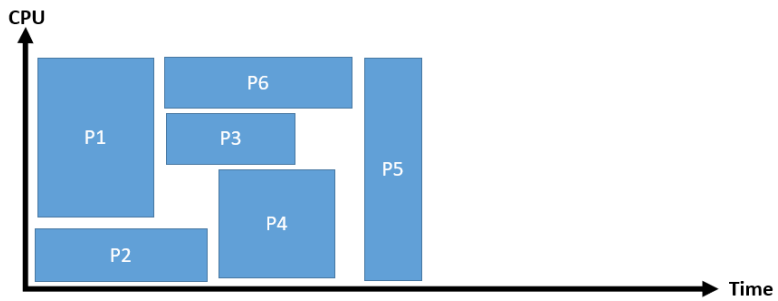
Definition of the problem - Example



Definition of the problem – Bad Example



Definition of the problem – Good Example



NP-Complete

NP-complete was a term first coined in the Cook-Levin theorem. It was used to define certain problems with a given complexity, problems which are known to be solvable using a *brute force* approach. While verifying the solution can be done fairly quickly.

Job Scheduling is a known NP-Complete problem (similar to the knapsack problem). It is solvable by *brute force* attempting all permutations of job scheduling and selecting the best overall solution. This is done in time complexity $O(n!)$.

Our hopes is not to find the *best* solution, rather to find a *good* enough solution. Which would still outperform and deterministic scheduling that is currently done.

	P	NP	NP-complete	NP-hard
Solvable in polynomial time	✓			
Solution verifiable in polynomial time	✓	✓	✓	
Reduces any NP problem in polynomial time			✓	✓

Background

Scheduling Algorithms

Here we will give a brief background about some basics scheduling algorithms. More “advanced” algorithms are a configuration of these basic algorithms in one form or another.

- **First Come First Serve (FCFS)** – allocating resources in the order in which the processes arrive
- **Shortest Job First (SJF)/Longest Job First (LJF)** – Sort the algorithms by length, and allocate resources either by longest or shortest time required.
- **Round Robin (RR)** – Allocates a set “time slot” for each process allowing the process to progress (use the resource) for the given time, after which it must relinquish the resource until its next cycle.
- **Priority Based Scheduling** – Organize the processes by a given priority metric, and allocating resources by importance.
- many more… (hybrids)

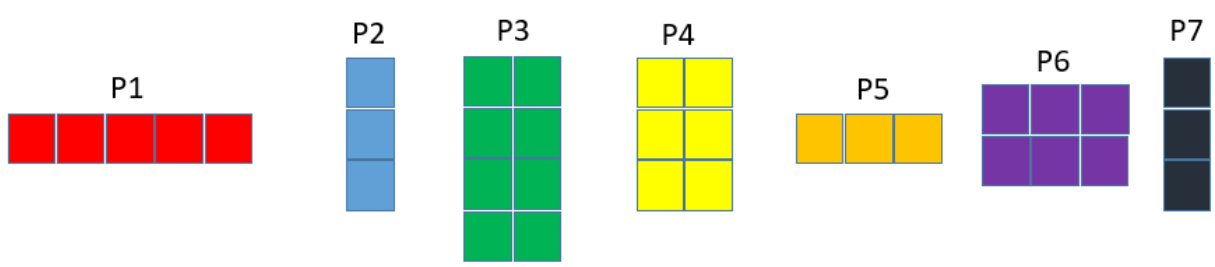
Each algorithm has its uses and systems depending on their requirements will implement the one that best suits its purpose.

Preemption vs. Non-Preemption – a preemptive algorithm grants the scheduler the capability of disallocating a resource to a given process, clear example would be RR. While a non-Preemptive algorithm would not permit the scheduler to remove a process from a resource. Preemption comes with a cost known as *context switch*, this is required overhead to store the state of the process being aborted, so that when it chooses to run again it may continue where it left off.

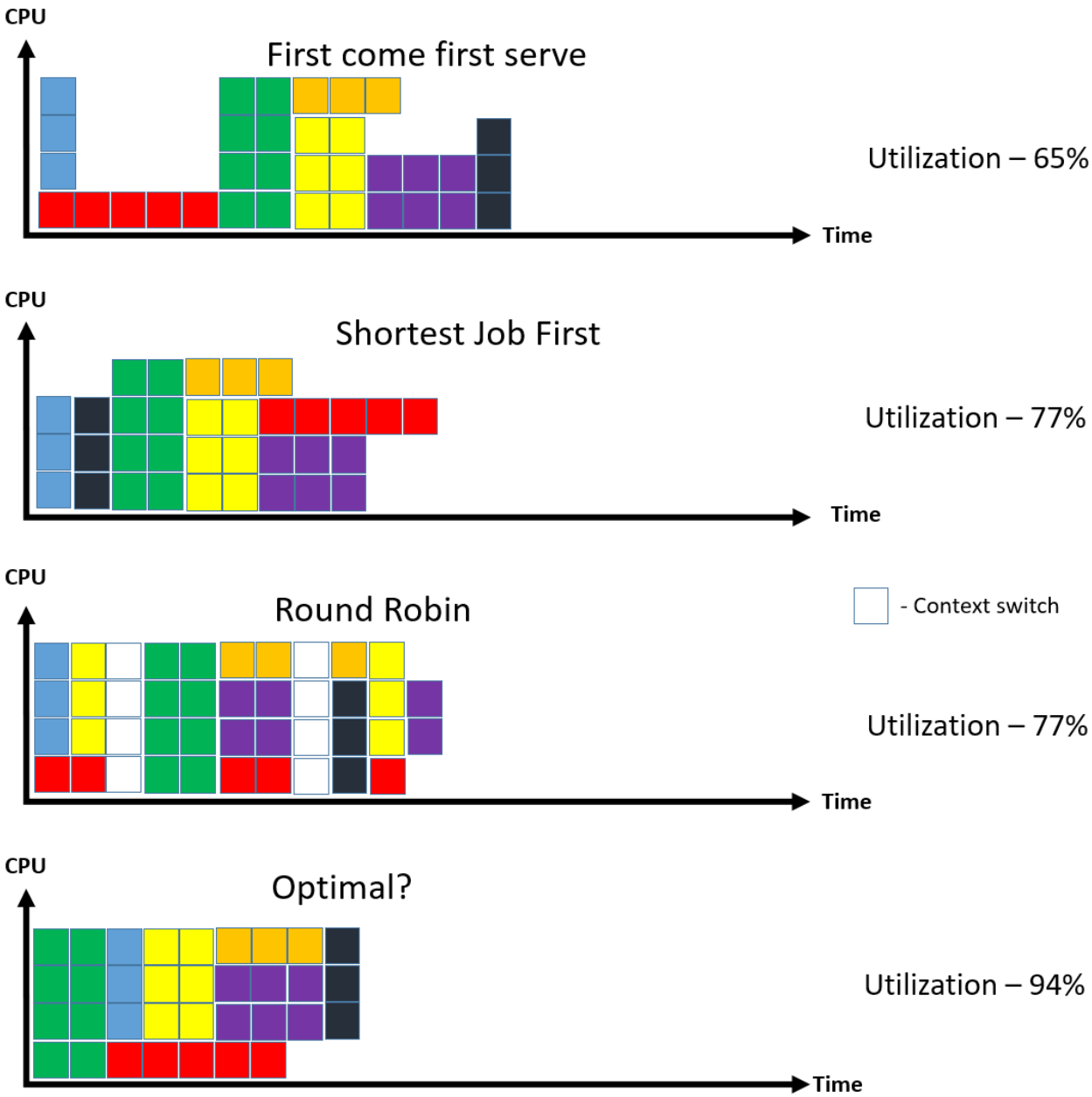
Comparing Algorithms Using Utilization Metric

Here we can show the different algorithms and how they would react to a set of given tasks. The resources available to the scheduler are 4 CPUs.

The order in which the tasks arrive with the amount of CPUs requested:



<u>How each algorithm would allocate the CPUs</u>



As we can see none of the deterministic algorithms reached the optimal implementation (which was done by hand), perhaps a more advanced

(Machine Learning driven algorithm) would have slotted the process in this manner?

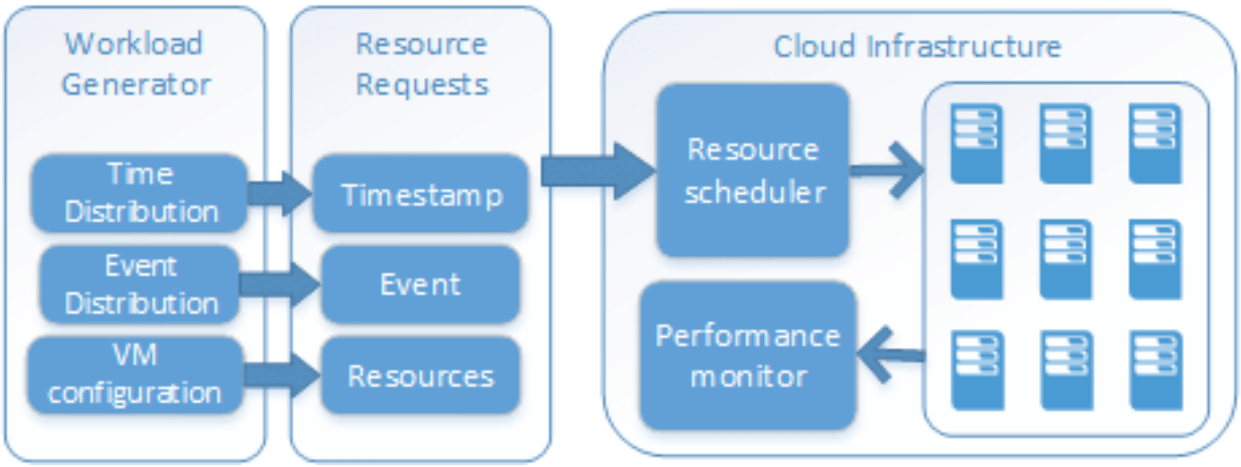
Cloud Scheduling

To help determine our problem we choose to confine our algorithm to a given system, in particular a cloud server. There are a few reasons we chose this, we find this to be a very sterile work field for which we can test the basics of our research. Another reason is we believe that this particular system can benefit the most from such a solution, understand the assumptions about this solution will explain why.

How does Cloud Work (abstract)

There exists three layers to cloud computing (in broad terms).

1. The first layer “Workload Generator” is the layer at which a client organizes the different process which he will request the server to execute, this includes packaging and configuring the data as required.
2. The second layer is again set by the client, defining the resources required to complete the given process. This completes the client side, this *package* is sent to the server
3. The final layer is the server implementation and execution of the *workload*, after this is completed the result are returned to the client



Cloud Computing Assumptions

Given this brief understanding of the dynamics of cloud computing we may draw several assumptions:

1. There is no requirement for preemptive scheduling. Given that no new tasks arrive during execution switching between tasks would cost valuable time on *context switches*. This a rather light assumption which can be easily integrated into any solution offered.
2. All tasks arrive together in the delivered *package*. We may sort the items without interruption as we see fit
3. To allow a clean environment we assume no user/hardware interrupts. No new tasks are introduced beyond those arrived in the package.

4. Finally that each task comes with a known run time requirement. This assumption is based on the fact that we expect the server to run reoccurrences of the same processes and this can be learned.

None of these assumptions are too constraining and can easily be rectified in a solution offered, although as a first step they allow for a very clean environment in which we can test our hypothesis.

Literary Survey

Through the project we conducted an intense literary survey. Part of survey was conducted to help us determine the right system in which environment we wished to run our simulation. In the following chapter we will present several notable scientific papers.

One thing to keep in mind that we came across during our survey was the fact that a majority of the publications recognized the problem as a complex one, and opted to use a reduced version of the problem. Granting variable assumptions that could be made for the given environment. No general practical solution was outlined in any publication they we found, they either covered the problem from a theoretical standpoint or solved it in a reduced environment.

Existing Solutions

Publication I - Implementing a Process Scheduler Using Neural Network Technology[4]

A Master thesis by Peter Bex

- Goes into great detail about different AI techniques as candidates to use to implement a Machine Learning-Based scheduler.
- Focuses only on **multimedia files** as tasks to be run. Focuses on a **uniprocessor**.
- Goes into much detail about the theoretical applications of such a solution to scheduling
- <u>Conclusion:</u> Bex found that selecting the “correct” features with which to learn, resulted in a large discrepancy between the outcomes. Though not for all tasks, he found that the select features had high correlation with specific tasks.

*Interestingly in his personal note he stated that when he began his research he thought the solution would be very straight forward and he completed his thesis feeling humbled by the complexity of the problem, we felt this to be very relatable.

Publication II - Applying Machine Learning Techniques to improve Linux Process Scheduling[5]

Atul Negi, *Senior Member, IEEE*, Kishore Kumar P.

- Focused not on changing the algorithm, but rather focused on implementing dynamic *time slices* granted to tasks through the use of Machine Learning techniques.
- Studied repetitive tasks

- Conclusion: using the metric *turn-around-time*, the authors found that using both *static and dynamic features* of given tasks, a modified scheduler was able to reduce the time by 1.4-5.8%

Publication III - A Machine Learning Approach for Improving Process Scheduling: A Survey[6]

Siddharth Dias1, Sidharth Naik2, Sreepraneeth K3, Sumedha Raman4, Namratha M

- This publication offered no solution to the problem but interestingly enough questioned if a Machine Learning Scheduler was a viable solution. The authors wished to determine whether pursuing such a solution had any potential
- In the grand scheme the authors determined that indeed many publications on the subject had promising results and found many cases where AI schedulers **outperformed** deterministic schedulers.
- Conclusion: A very profound conclusion from surveying multiple articles the authors determined that the most promising approaches, were the ones focused on discovering different features for given tasks (this will come up in our conclusion section)

Our Research

Defining the Problem

Now that we have laid out the foundations to understanding scheduling and metrics by which we can measure them against each other. We will readdress the original question we set out to answer framed using common terminology from the previous chapters.

Research Question

Can we improve on current scheduling algorithms using ML Technology?

Metrics of Measurement: CPU Utilization – the proportion of time used to progress tasks over the overall CPU runtime until tasks batch completion.

Definition of the Problem: For a given workload (series of tasks) will our ML algorithm be able to outperform current heuristic scheduling algorithms?

Research Timeline

Our original intent when posing this question was to attempt to use Machine Learning Techniques to single out certain features of processes. Which if given priority based on these features would help the scheduler efficiency.

Realizing this was a large leap forward, we took a step back wondering if advancing in this direction was a good idea. We did not want to lose focus of the main objective of the project, we instead decided to try and come up with what we though to be logical features and have the algorithm learn based on the given features.

Second obstacle we encountered was running our algorithm in a live system. First attempts were not successful. Measurements within kernel is a project of in on its own, this was a main struggle as we could not say with confidence that the measurements were not skewed due to noise (interrupts, power faults, memory allocation, etc.).

We will come back to these two points in the conclusion section.

Given these two obstacles we decided on creating a simulation of a scheduler. This allows the freedom to pursue the Machine Learning aspect of our question (the main topic of our research).

Model Version 1

Simulator Characteristics

Abstract

This version of the simulator enabled as much configuration as possible, using different flags we allowed ran two main modes. Machine Learning mode, and heuristic algorithm mode, the intent was to be able to get similar output metrics in both scenarios thus allowing a level comparison.

The Machine Learning algorithm was a regression problem, trying to maximize the utilization across multiple cores for a given workload. The heuristic mode simply arranged the tasks as they would in a live system.

Regression Model – Defined for models with a continuous output $f: R \rightarrow [a,b]$. For a given loss function, the algorithm attempts to minimize the loss.

Input: Workload, List of Start Times

Output: Utilization, Throughput, Latency, Avg Expansion Time, Avg Wait Time, Response Time, Avg Termination Time

Machine Learning Technique: A supervised - regression problem

Algorithm Model: Fully Connected Neural Network

Metric: Minimization of $f(input) = 1 - utilization(norm)$

$$f(W): R \rightarrow [0,1]$$

Model Definition

Neural Network Design (fully Connected)

Input Layer: $2(RT, BW) \times 8(Tasks) = 16\, Neurons$

Hidden Layer: $64\, Neurons, activation: Self$

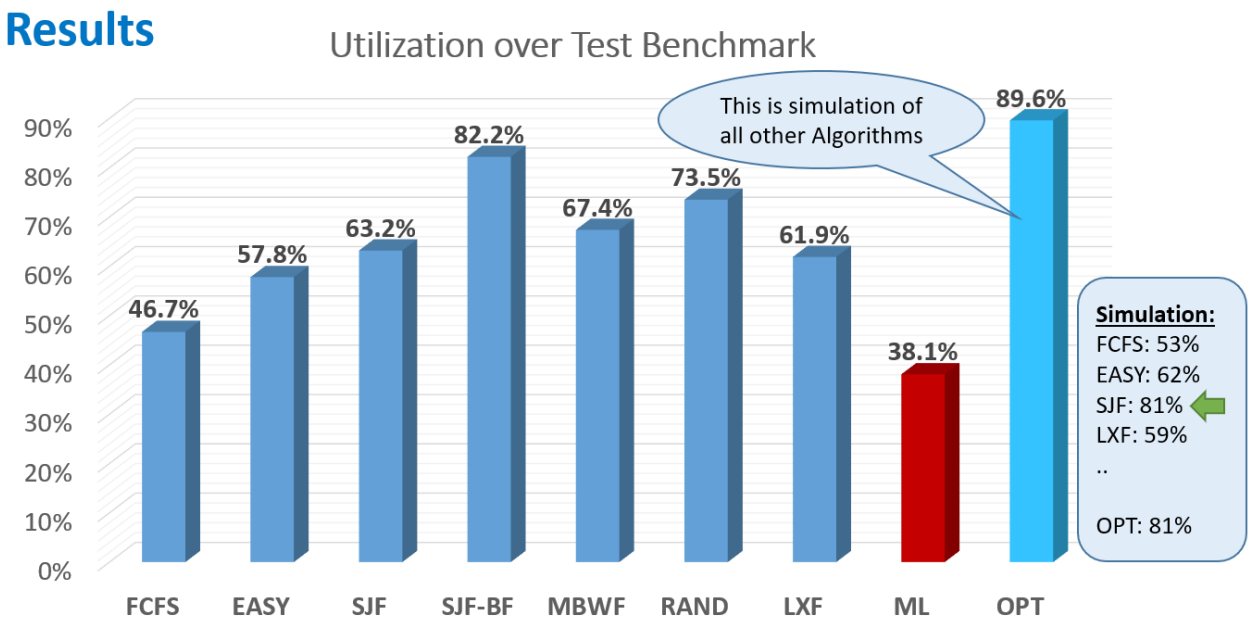
Output Layer: $8\, Neurons, activation: Self$

Model

- Too Slow ($H\ each\ epoch!$)
  Using different data analysis tools was the main reason each epoch took so long (reading from certain file types for example)
- The algorithm had difficulty fitting “valid” schedules
  Only 41.3% of the output data was valid
- Average Utilization: **38.1%**

Results

Compared with performance done by heuristic algorithms:



*The far right cyan bar, is the overall performance if for each workload we processed the “ideal” heuristic algorithm to run. The overall performance is 89.6%

Summary

The simulation required more fine tuning, an unacceptably high rate of illegal schedule inlays were created by the simulation. Moreover, when comparing the *utilization* of our algorithm compared with heuristic known algorithms. Ours under preformed by a large margin.

Model Version 2

Simulator Characteristics

Abstract

After drawing conclusions from the previous version, we decided the second version first and foremost needed to create more legal outputs. We changed this by implementing “semi-manual” outputs (explained in more depth under the features category). Further more singling out the cumbersome dealing with reading/writing to certain data files which we identified as being a main source of time consumption, we reorganized how the data was processed, which libraries to use, etc. Both these decisions lead to our first significant breakthrough in the research. This model could now grant us insight into further applicable finetuning to answering the research question.

Here too the model attempted to solve a Regression problem, for the loss function given utilization as metric to optimize.

Added Features

- Semi-manual scheduling
  The Simulator no longer attempted to grant time slots for each task. Rather the output was a an ordering for which to run the tasks. In this case where consecutive tasks could run simultaneous, due to the availability of CPUs, they were expected to be “manually” placed in the same time slot.
- Allowing simulator to select single algorithm implementation
- Simulator grants the user the ability to define which metric to optimize
- Hyperparameter Schedule

<u>Input</u>: Workload

<u>Output</u>: List of start times for tasks

<u>Machine Learning Technique</u>: A supervised - regression problem

<u>Algorithm Model</u>: Fully Connected Neural Network

<u>Metric</u>: Minimization of $f(input)=1-utilization(norm)$

$$f(W): R \rightarrow [0,1]$$

Model Definition

Neural Network Design (fully Connected)

Input Layer: $2(RT, BW) \times 8(Tasks) = 16\, Neurons$

Hidden Layer 1: $64\, Neurons, activation: ReLU$

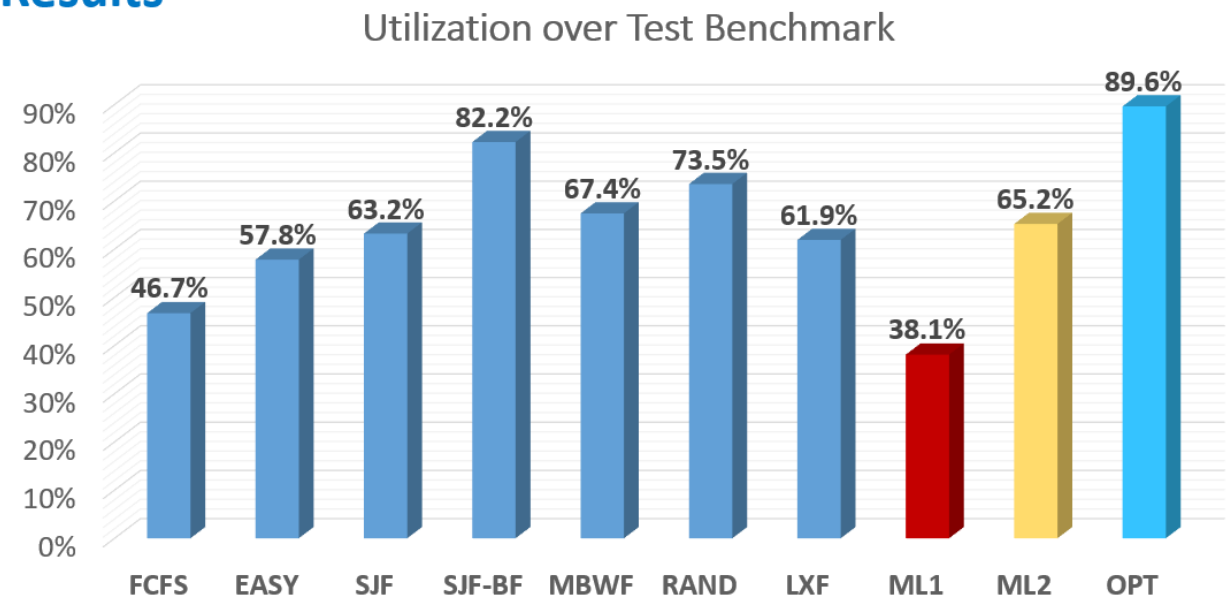
Hidden Layer 2: $32\, Neurons, activation: ReLU$

Output Layer: $8\, Neurons, activation: Sigmoid$

Model Results

- Improvement on prediction time, now only takes several minutes
- Improvement to legal outputs, all outputs where now legal (100%)
- Average Utilization: **65.2%**

Results



Summary

We can see a huge leap in utilization metric when compared with our previous model. All of our outputs now were legal outputs. Yet still heuristic algorithms outperformed our model, we still could not outperform known algorithms. This lead us to attempt a third and final model.

Model Version 3

Simulator Characteristics

Abstract

This is the final model we developed and the one we are most satisfied with. After spending some time trying to think of ways to improve our algorithm (we found different approaches to be of little gain), we realized the mark we were marking for (Opt bar) is a perfect candidate. If we could for any workload always select the “ideal” heuristic algorithm we can guarantee to be at least on par with known algorithms for a given workload, and in most cases outperform a system running a specific algorithm.

This gave birth to the classification model we created. This model no longer attempted to schedule the tasks, rather to predict which heuristic model would be best suited for a given workload.

Simulator Features

- Multi-mode added (allowing multiple different algorithms to run for comparison)
- Made changes to pre-processing of data
- Run time was optimized (each epoch takes $5m$)

Input: Workload

Output: Optimal Heuristic Algorithm to use, a vector with the selected algorithm light by a 1 all other algorithms show 0.

Machine Learning Technique: A supervised - **Classification** problem

Algorithm Model: Fully Connected Neural Network

Metric: Minimization of $f(input)=1-utilization(norm)$

$$f(W):V \rightarrow [0,\ldots,1,\ldots,0]$$

Model Definition

Neural Network Design (fully Connected)

Input Layer: $2(RT, BW) \times 8(Tasks) = 16\, Neurons$

Hidden Layer 1: $256\, Neurons, activation: ReLU$

Hidden Layer 2: $128\, Neurons, activation: ReLU$

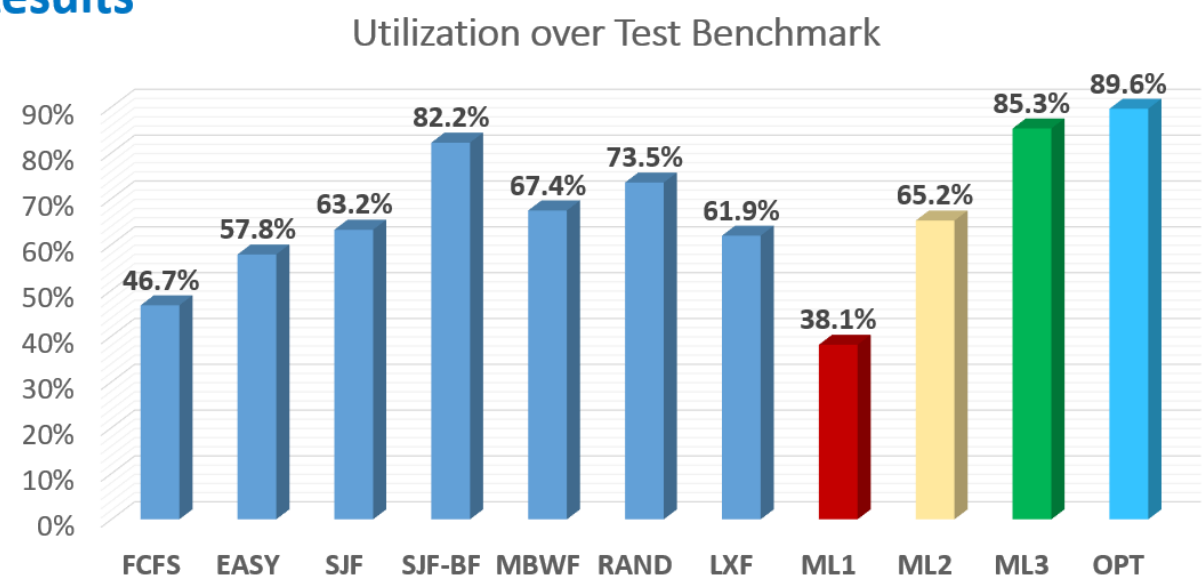
Hidden Layer 3: $64\, Neurons, activation: ReLU$

Output Layer: $8\, Neurons, activation: Softmax$

Model Results

- Improvement on prediction time
- Selected an algorithm that would deliver best utilization
- Average Utilization: **85.3%**

Results



Summary

We are very proud of these results. We can see that each iteration of the model resulted in an increase in utilization with a final pinnacle outperforming all other algorithms. By reassessing the parameters of our problem we found an more fitting solution than we initial set out to test (classification vs. regression). The discrepancy between our algorithm and the cyan bar, we aimed for, is explained that our model for some workloads miss classified, within an acceptable rate ($miss\, rate = 4.3\% < 5\%$)

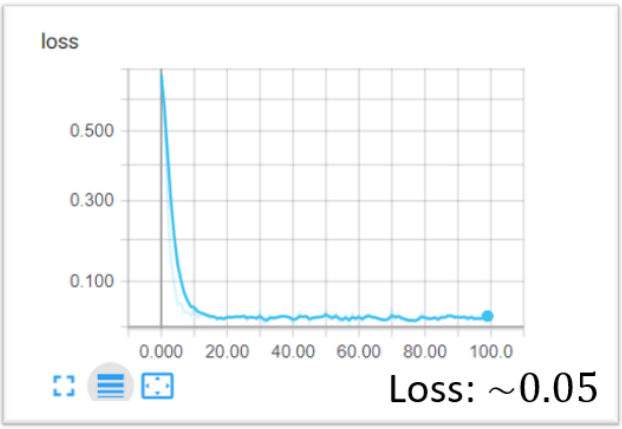
Having constructed this model we felt our research for the initial stage (showing proof of concept) had come to a positive conclusion.

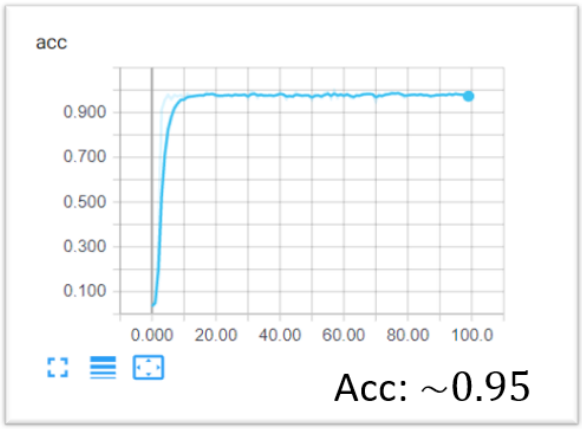
Further Testing

Now we began to display our selected model and cast it through a series of different tests.

Here we display how compared against our target in aspects of loss and accuracy:

Loss and Accuracy





$$ML = 85.3 \approx 89.6(OPT) \cdot 0.95(ACC)$$

The following series of test were conducted using different types of tests.

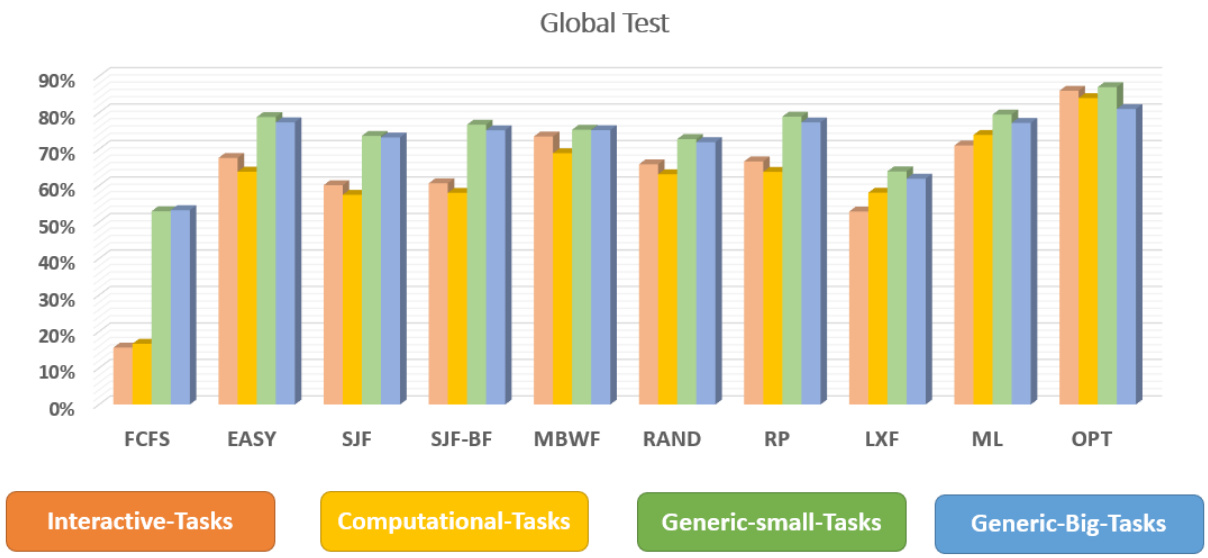
Interactive-Tasks: A batch of tasks requiring short bursts of CPU time to run (number of CPUs is random)

Computational-Tasks: Opposed to interactive, here the tasks require long stretches of CPU run time (while the number of CPUs are random)

Generic-Small-Tasks: Here the workloads were composed of small tasks requiring a small number of cores with a small amount of run time to complete

Generic-Big-Tasks: The tasks in these workloads required multiple cores and a long length of time to process each task.

Global Test



[10,000 examples, each test]

For a cleaner comparison:

Global Test

Test name	Max RT	Max BW	Size	FCFS	EASY	SJF	SJF-BF	MBWF	RAND	RP	LXF	ML	OPT
Generic - small	4	4	10K	15.6%	67.6%	60.1%	60.7%	**73.5%**	65.8%	66.6%	52.9%	71.0%	86.0%
Computation	16	4	10K	16.7%	63.8%	57.5%	58.0%	68.9%	63.1%	63.8%	58.1%	**73.9%**	84.0%
Interactive	4	16	10K	53.0%	78.8%	73.6%	76.7%	75.3%	72.8%	78.9%	63.9%	**79.5%**	87.0%
Generic - big	16	16	10K	53.3%	**77.4%**	73.2%	75.2%	75.2%	71.9%	77.3%	61.9%	77.1%	81.0%

Not bad though

In both Computation and Interactive tasks our Model outperformed all existing algorithms. While for the Generic sets our algorithm came a close (trailing less than 3%) place behind the leading candidates.

Inference

- For a workload of n tasks given, classification by our final model took $O(n)$ time to predict
- Any algorithm at best requires $O(n)$ time to complete
- Conclusion: predicting which algorithm to run using Machine Learning, does not change overall Big O, time complexity to change.

Conclusions

- For the average case, our NN outperformed heuristic algorithms
- We found that the NN approach is best suited for workloads with distinctive patterns
- Now that proof of concept has been established, there are many follow up avenues that can explored, regarding NN driven Scheduling

What's Next?

This project has left us with more of an appetite than before, there is a lot of potential to developed in this area.

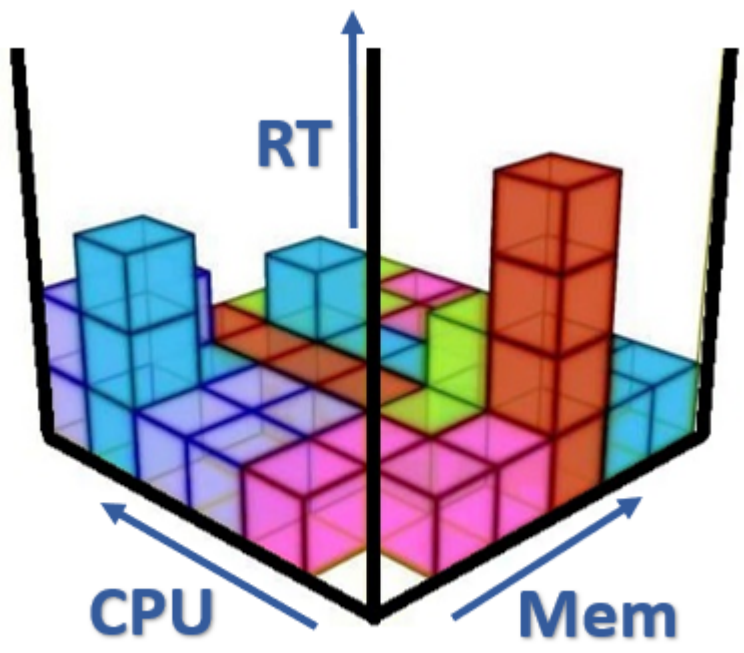
When we first set the question, we were curious if we could use machine learning to discover, previously underrated parameters by which to schedule, which would increase utilization. Due to the constraints discussed in this book we oriented the question in a different direction, with the attempt to show proof of concept of machine learning as a viable approach to scheduling.

This being said and concluding that indeed this has merit, we wish to pivot back to our original idea.

Multi Parameter

Using several indicators for optimal scheduling

- User/Kernel space
- Memory usage
- Real time constraints (i.e. networking)
- GPU vs CPU usage
- and many more…



The second field we wish to proceed to, is **live systems**. This is a bit more tricky are will require a steep learning curve on the system we will eventually choose (most likely servers). Acquiring measurements within the operating system is on it's on a difficult tasks.

But this would be the true practical usage of anything we discover.

Using such predictions could help us select the proper server on which to run a batch of tasks for instance.

Multi parameter - Amazon SageMaker

Instance Type	vCPU	GPU	Mem (GiB)	Mem GPU (GiB)	Network Performance
ml.p3.2xlarge	8	1xV100	61	16	Up to 10 Gbps
ml.p3.8xlarge	32	4xV100	244	64	10 Gigabit
ml.p3.16xlarge	64	8xV100	488	128	25 Gigabit
ml.p3dn.24xlarge	96	8xV100	768	256	100 Gigabit
ml.p2.xlarge	4	1xK80	61	12	High
ml.p2.8xlarge	32	8xK80	488	96	10 Gigabit
ml.p2.16xlarge	64	16xK80	732	192	25 Gigabit
ml.g4dn.xlarge	4	1xT4	16	16	Up to 25 Gbps
ml.g4dn.8xlarge	32	1xT4	128	16	50 Gbps
ml.g4dn.12xlarge	48	4xT4	192	64	50 Gbps
ml.g4dn.16xlarge	64	1xT4	256	16	50 Gbps

References

[1] https://royalsocietypublishing.org/doi/10.1098/rsta.2019.0061
[2] ROUND ROBIN SCHEDULING FOR FAIR FLOW CONTROL IN DATA COMMUNICATION NETWORKS (1986)
[3] David L. Parnas, “Software Aging”, Communications Research Laboratory, McMaster University
[4] See **Implementing a Process Scheduler Using Neural Network Technology** in git repository for further reference
[5] See **Applying Machine Learning Techniques to improve Linux Process Scheduling** in git repository for further reference
[6] See **A Machine Learning Approach for Improving Process Scheduling: A Survey** in git repository for further reference