MACHINE LEARNING-BASED LINUX KERNEL HANDLER

Students: Netanel Rothschild, Roy Maor Lotan

Supervisor: Dr. Roman Kaplan

Semester: Spring 2021

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.

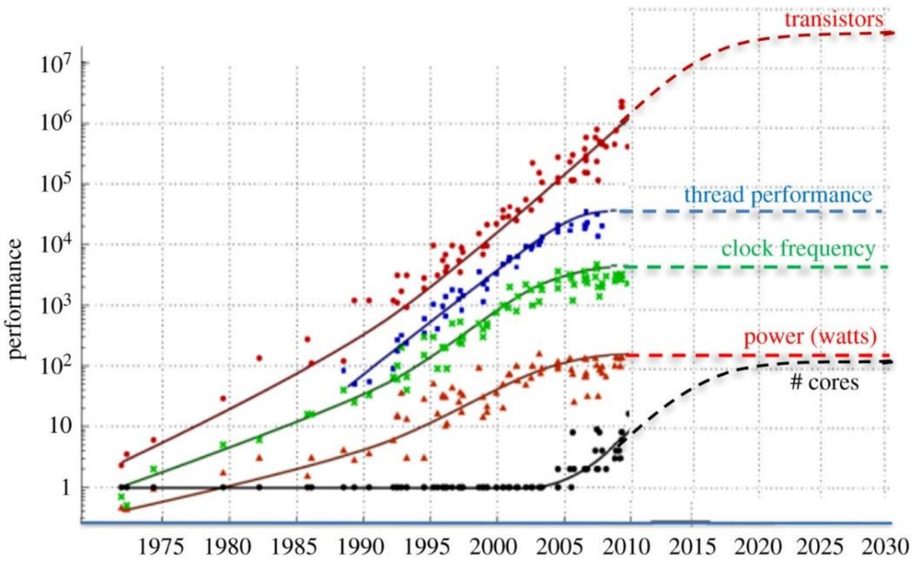
The famous Moore's law predicted that the number of transistors on a microchip doubles every two years, it is famous because over the past fifty years it turned out to be accurate. A definite example of the progress we've made is the sheer fact that more than half the people in the world walk around 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.

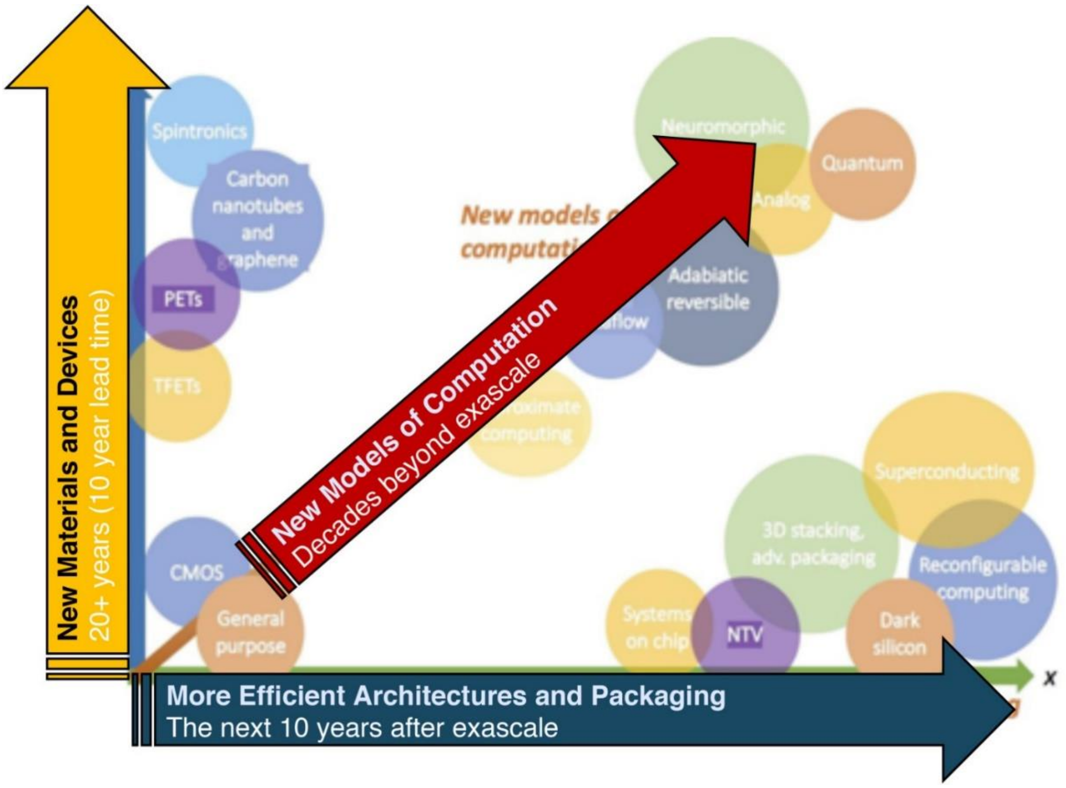
Throughout our degree we studied multiple technologies, some seemingly archaic, some very modern. Noticing this discrepancy we began to question, could this uneven playing field cause problems as the gap grows?

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.


New Materials and Devices
20+ years (10 year lead time)
Spintronics
Carbon nanotubes and graphene
PETs
TFETs
CMOS
General purpose
New models of computation
New Models of Computation
Decades beyond exascale
Neuromorphic
Quantum
Analog
Adabiatic reversible
Dataflow
Approximate computing
Superconducting
3D stacking, adv. packaging
Reconfigurable computing
Systems on chip
NTV
Dark silicon
More Efficient Architectures and Packaging
The next 10 years after exascale
x

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

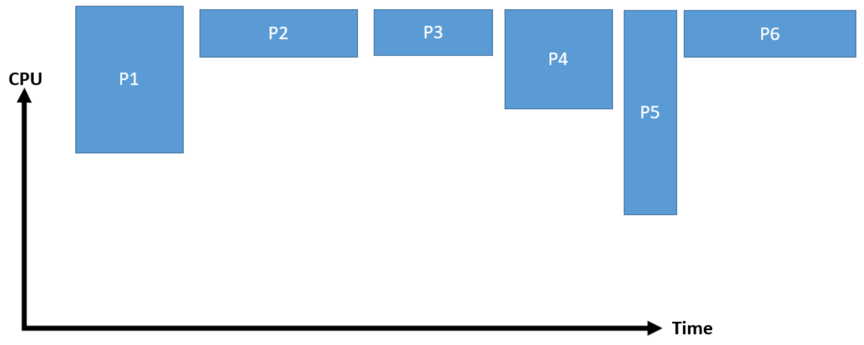
There are many ways to measure the performance of a scheduler:

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"

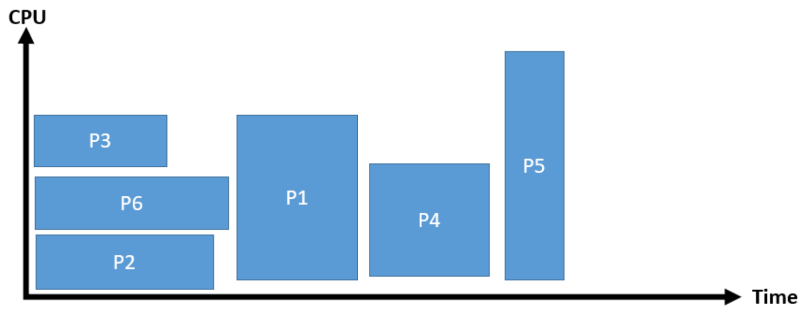
5. **Response Time** – Amount of time it takes from when a request to run has been submitted until a first response is produced.

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

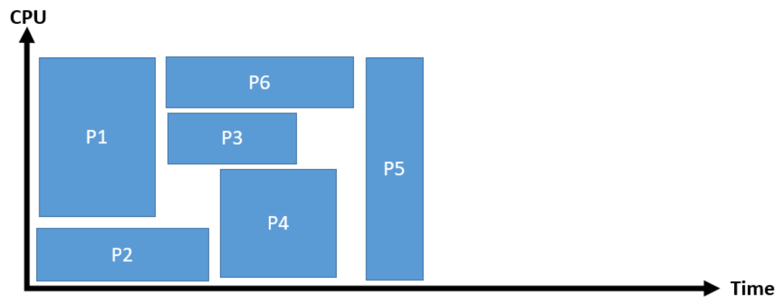
Definition of the problem - Example



Definition of the problem – Bad Example



Definition of the problem – Good Example

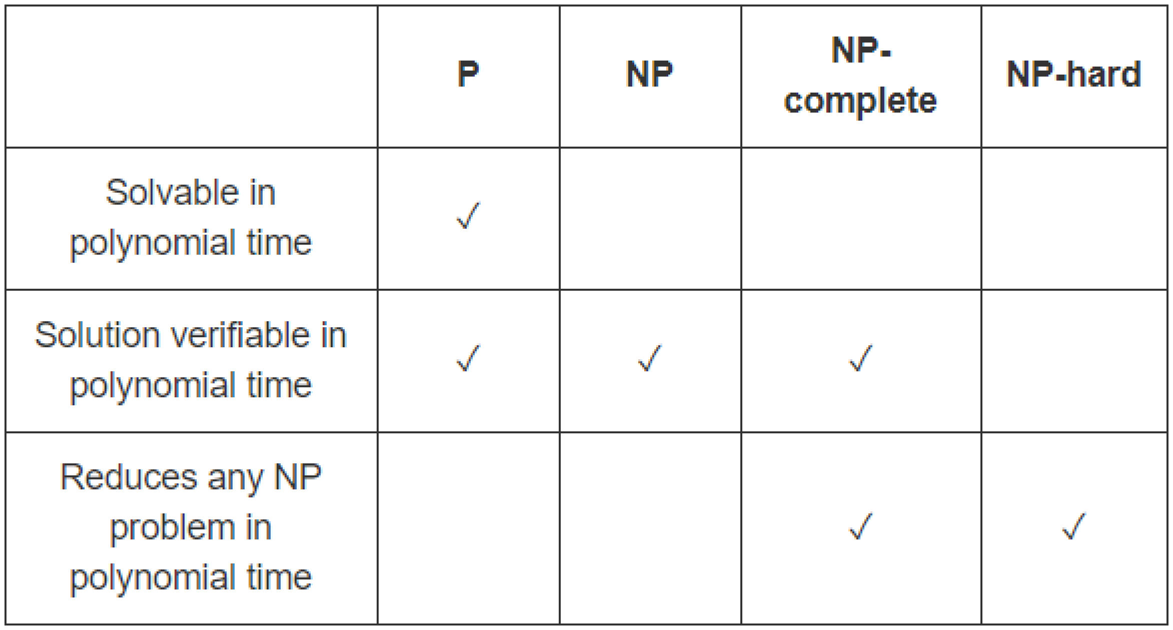


NP-Complete

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)

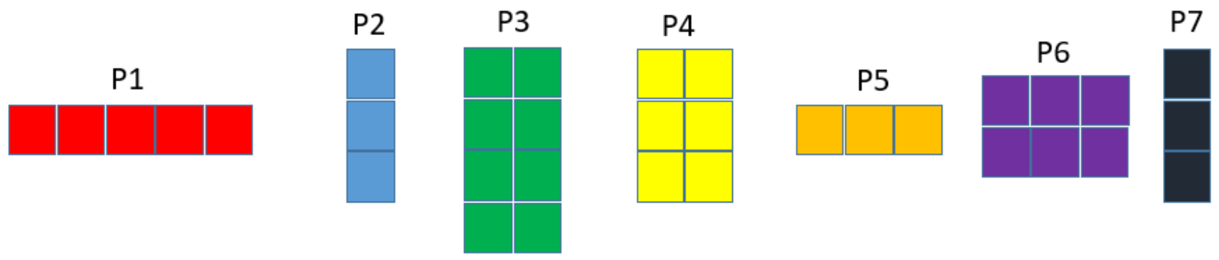
Each algorithm has its uses and systems depending on their requirements will implement the one that best suits its purpose.

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.

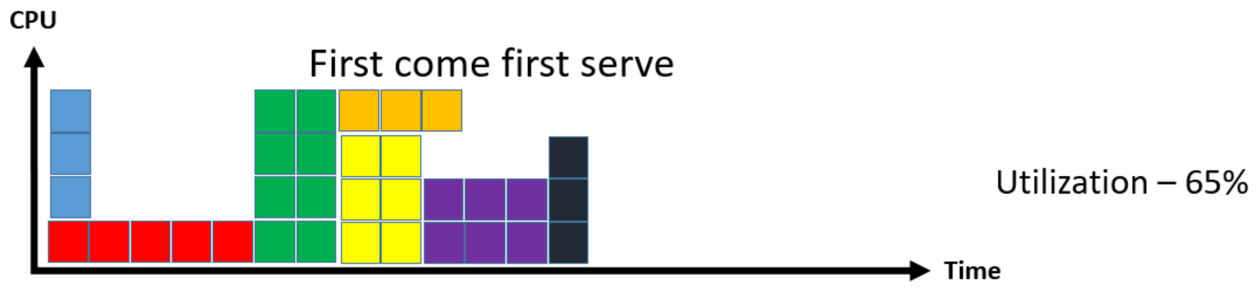
Comparing Algorithms Using Utilization Metric

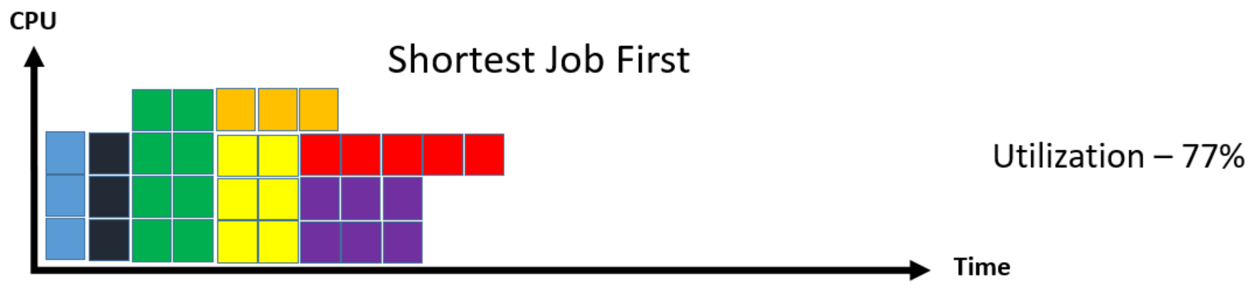
Here we can show the different algorithms and how they would react to a set of given tasks. The resources available to the scheduler are 4 CPUs.

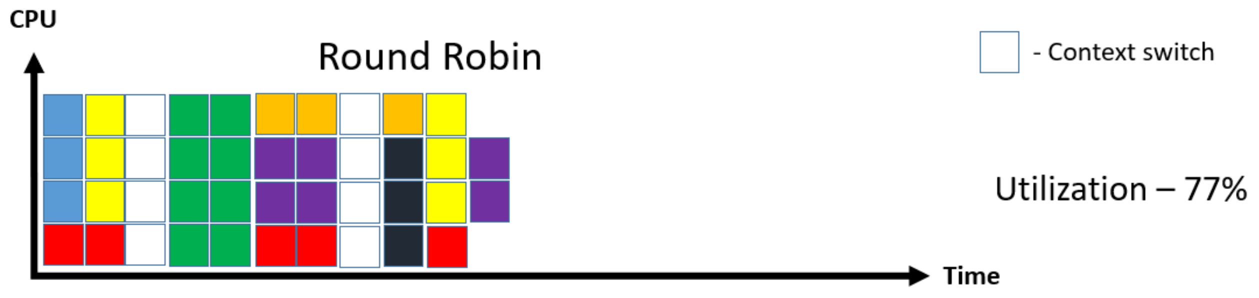
The order in which the tasks arrive with the amount of CPUs requested:

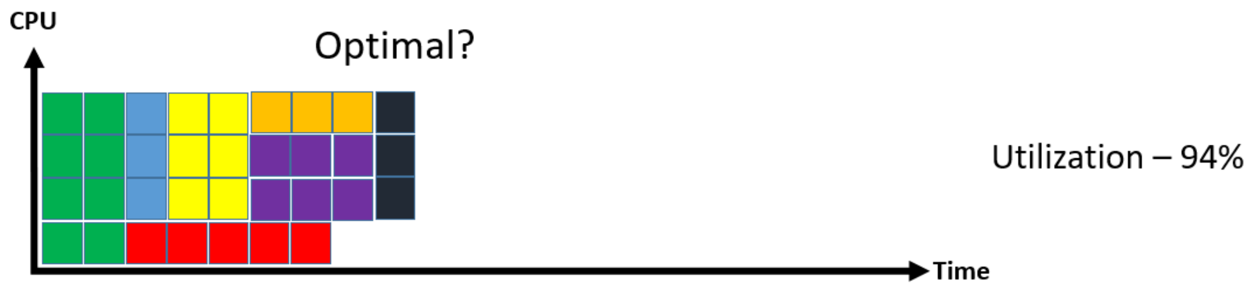


<u>How each algorithm would allocate the CPUs</u>









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?

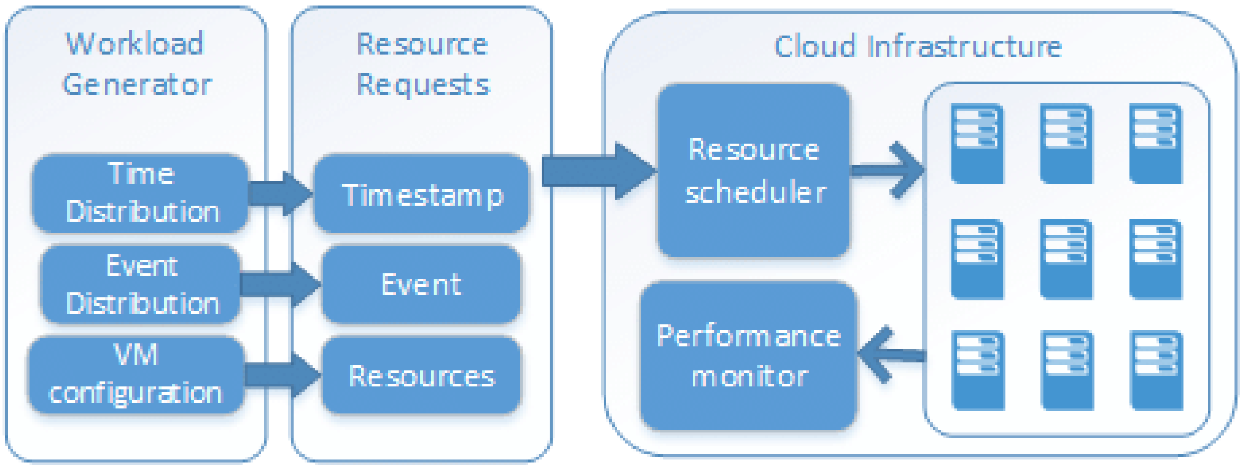
Cloud Scheduling

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
- Conclusion: 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

Simulation – proof of consept

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 $f: \mathbb{R} \to [a, b]$. 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 $f(input) = 1 - utilization(norm)$

$$f(W): \mathbb{R} \to [0,1]$$

Model Definition

Neural Network Design (fully Connected)

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

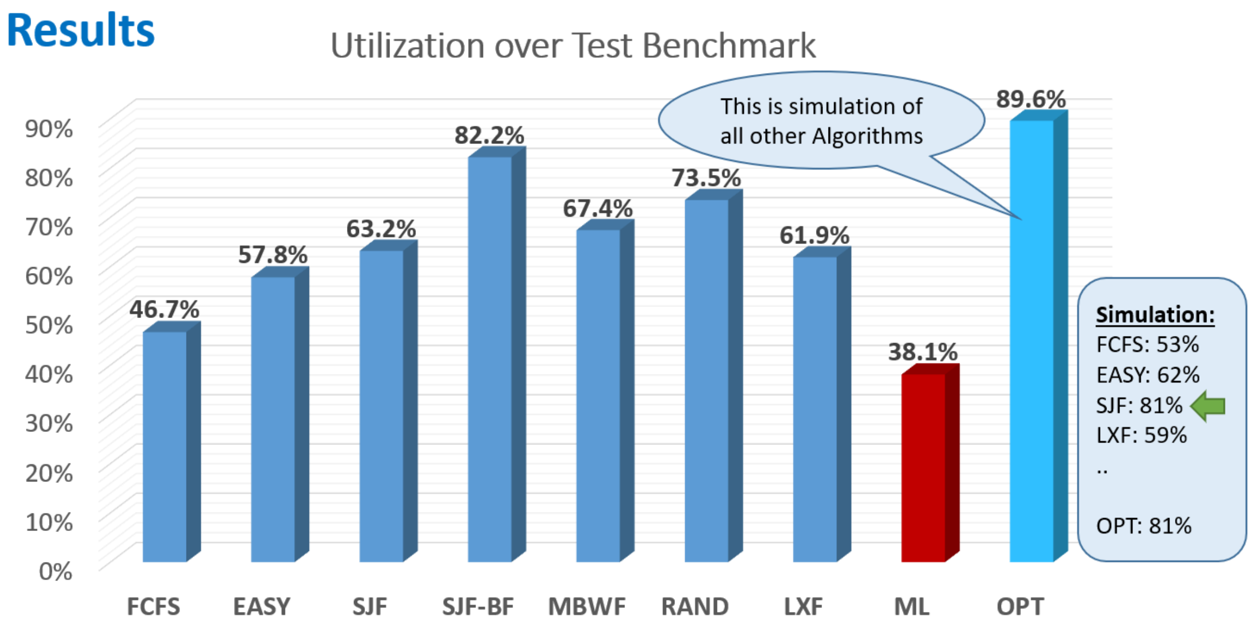
Hidden Layer: $64\ Neurons, activation: Self$

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

Model Results

- Too Slow ($\sim H\ each\ epoch!$)
  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%**

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.

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 $f(input) = 1 - utilization(norm)$

$$f(W): \mathbb{R} \to [0,1]$$

Model Definition

Neural Network Design (fully Connected)

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

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

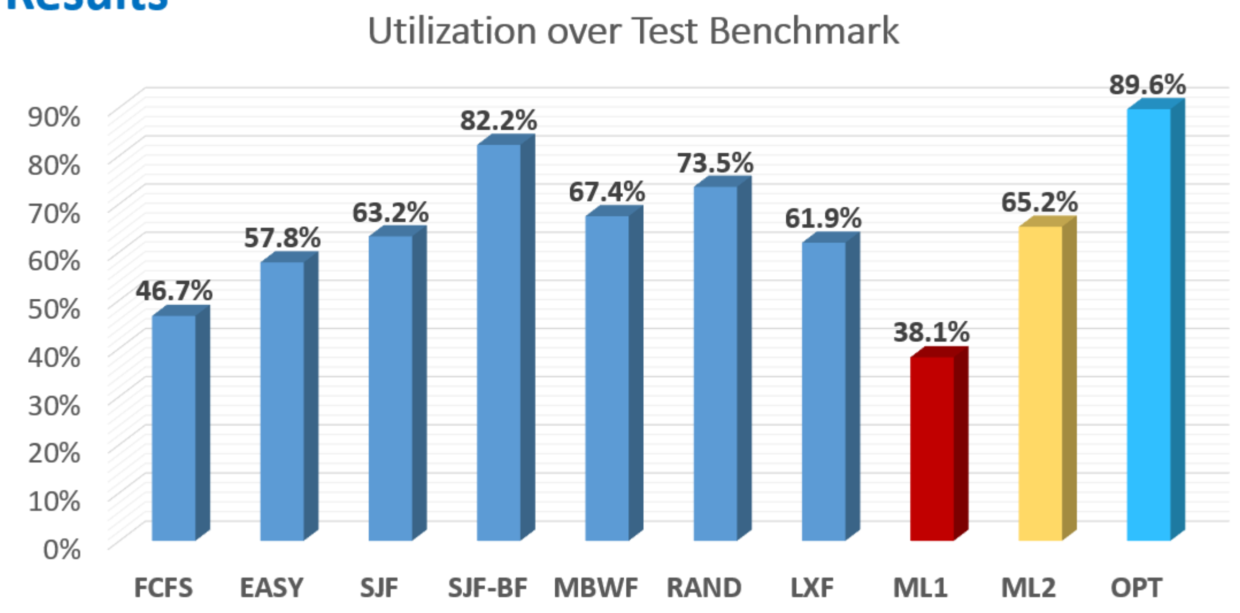
Hidden Layer 2: $32\ Neurons, activation: ReLU$

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 $\sim 5\ m$)

<u>Input</u>: Workload

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

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

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

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

$$f(W): \mathbb{V} \rightarrow [0, \dots, 1, \dots, 0]$$

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: $128\ Neurons, activation: ReLU$

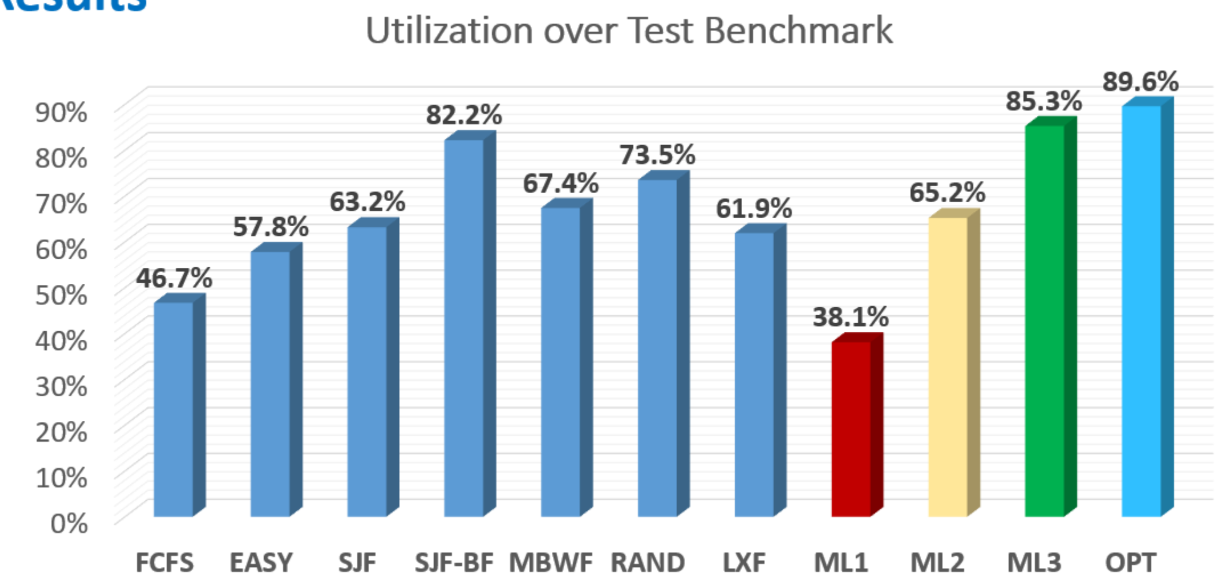
Hidden Layer 3: $64\ Neurons, activation: ReLU$

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 ($miss\ rate = 4.3\% < 5\%$)

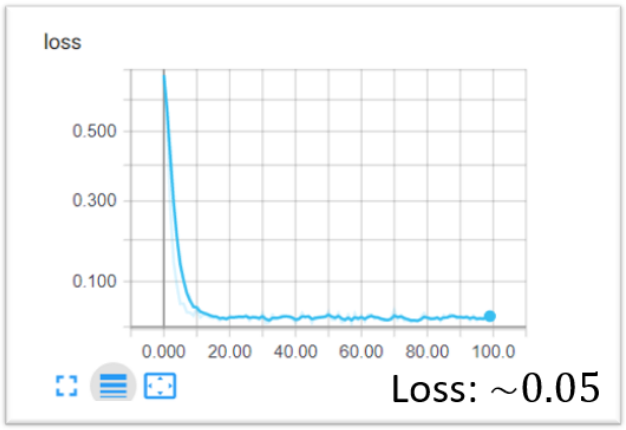
Having constructed this model we felt our research for the initial stage (showing proof of concept) had come to a positive conclusion.

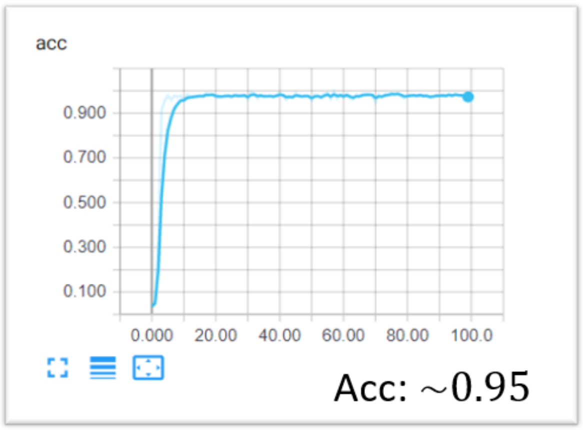
Further Testing

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.

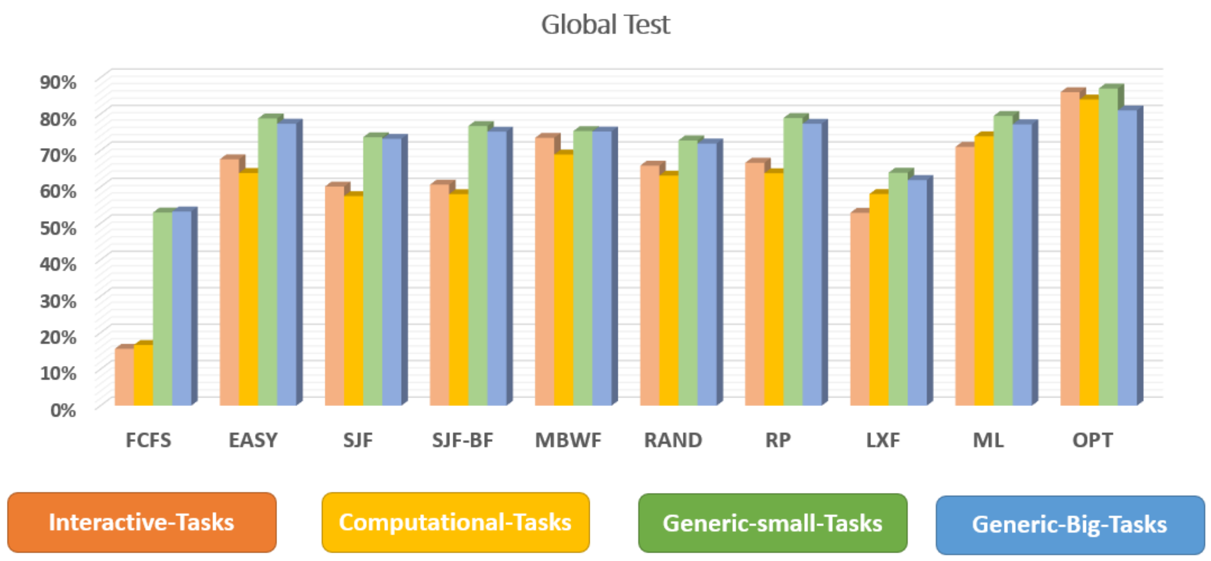
Interactive-Tasks: A batch of tasks requiring short bursts of CPU time to run (number of CPUs is random)

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

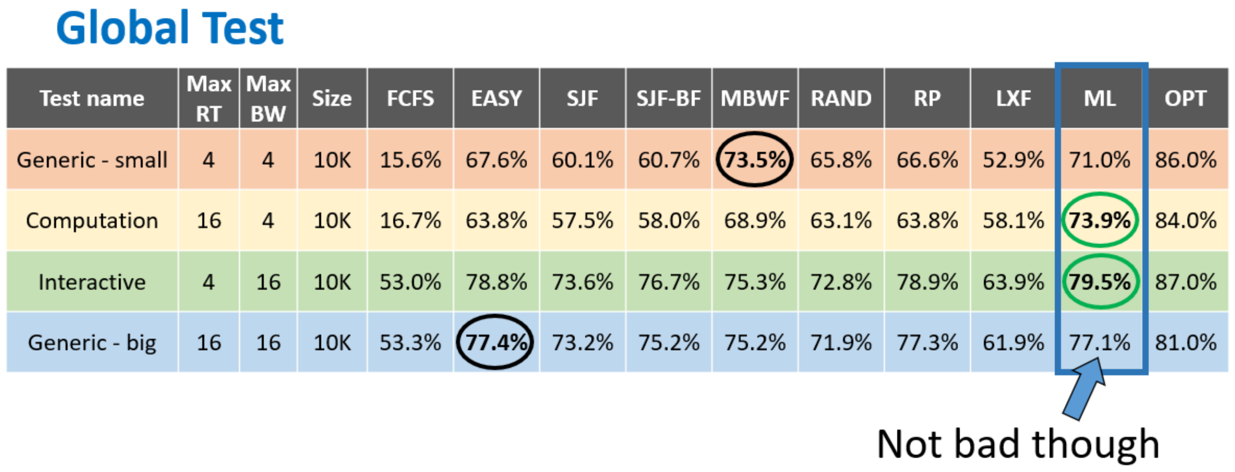
Generic-Big-Tasks: The tasks in these workloads required multiple cores and a long length of time to process each task.

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

Next Step

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

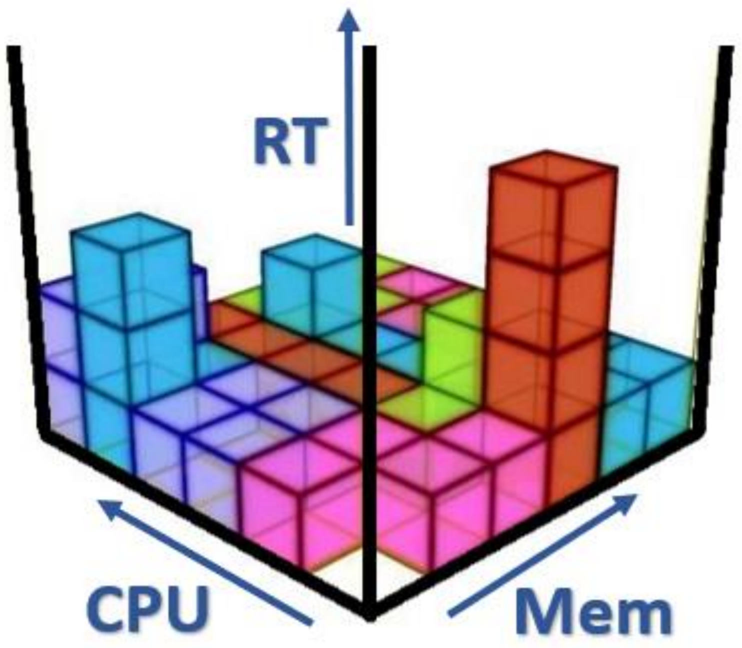
When we first set the question, we were curious if we could use machine learning to discover, previously underrated parameters by which to schedule, which would increase utilization. Due to the constraints discussed in this book we oriented the question in a different direction, with the attempt to show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

aws
AWS S3
AWS EC2
AWS RDS
Microsoft Azure
Azure Blob Storage
Azure Virtual Machines
Azure SQL Database
Google Cloud
Google Cloud Storage
Google Compute Engine
Google Cloud SQL

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

Cache predictor

Defining the Problem

As framed in the previous parts of the book, solving the memory fetch/miss problem may be the next major breakthrough, in computer computation power. The resources wasted on incorrect predictions are multiple fold and create huge backlogs for the scheduler. When correct predictions are made many of the delays can be averted.

In the first part of the project, we found it difficult to siphon data from the system. Working within time constraints we opted to show a proof of concept first, paving the way for when we are able to extract data. This brings us to the second part of the project.

We identified the cache as a core issue in the data retrieval streamline. The difference between data on hand in the cache as opposed to data on remote non-volatile memory was of multiple orders of magnitude different. Believing the cache to be a prime area in which to test our hypothesis (the one we determined in part one of this project), we selected it to be used.

The cache system has already a rather simple algorithm implemented, which is used. The existing algorithm is very straightforward and deterministic.

The following diagram shows the layout of the operations conducted by the cache.

1) Initially two processes (S1 and S2) have permission to access the same page in the main memory (colored green), the number in the memory area, is the number of pointers currently pointing at the page.

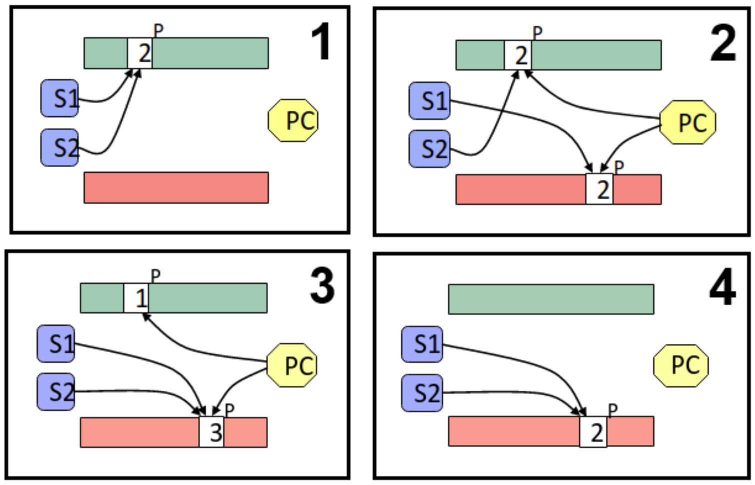
2) The eviction protocol begins. PC the page cache (an entity which exists solely for the purpose of handling on hand page retrieval/removal) holds pointers to the page in main memory and remote memory. S1, the first process to recognize its page has begun the eviction protocol, updates its pointer to the remote memory. We end the initial step eviction protocol in a state where the page is stored in main memory and in remote memory, with each entity of the page being pointed at by two pointers.

3) The protocol continues and it is process S2's turn to reassign it's pointer. The process S2 now updates its info to the fact the page is no longer on hand. When this step is complete the only pointer held to the main memory page is held by the Page Cache. In this scenario we can safely remove the page from main memory as no process will access the page directly from the memory, rather it will request that the page be brought to main memory from the remote memory. Currently the page in main memory has a single pointer pointing at it, with the page in remote memory being pointed at by three pointers.

4) The final step it is removing the pointers from the Page Cache as the pages are no longer on hand and all processes associated with the page know where the page is located and may call for it to be brought at any time.

Cache operation

- Page Cache
- Main Memory
- Remote Mem



We attempted multiple ways to get the data we desired, which will be given in detail in the following sub-chapters. For our model to predict correctly which data from memory may be removed and which is advisable to not have removed, we require a complete picture of the data usage. Along with processes behavior, and the operating system's behavior, there may be other hidden constraints we are unaware of. This led to the conclusion that we must have a complete picture of the cache.

Without an accurate picture of the cache, we could not guarantee consistent results.

Data Extraction

In the previous chapter we explained the importance of the data extraction process. If the data is incomplete or collected incorrectly, we risk training and predicting everything incorrectly, and we may not even know it.

Through the second half of the project, we found this to be the greatest challenge we encountered on our attempt to predict memory usage. The machine learning models turned out to be the more easier part of the project. The data siphoning, which was a main obstacle in the first part of the project turned out to be a major operation, following down paths we weren’t sure would lead to results, and in the end turned out to be futile attempts. Though we learned a lot in the process.

After each attempt we found ourselves heavily invested in the idea and found it difficult to convince ourselves that the approach was not the correct one.

The three attempts we made for data extraction were:

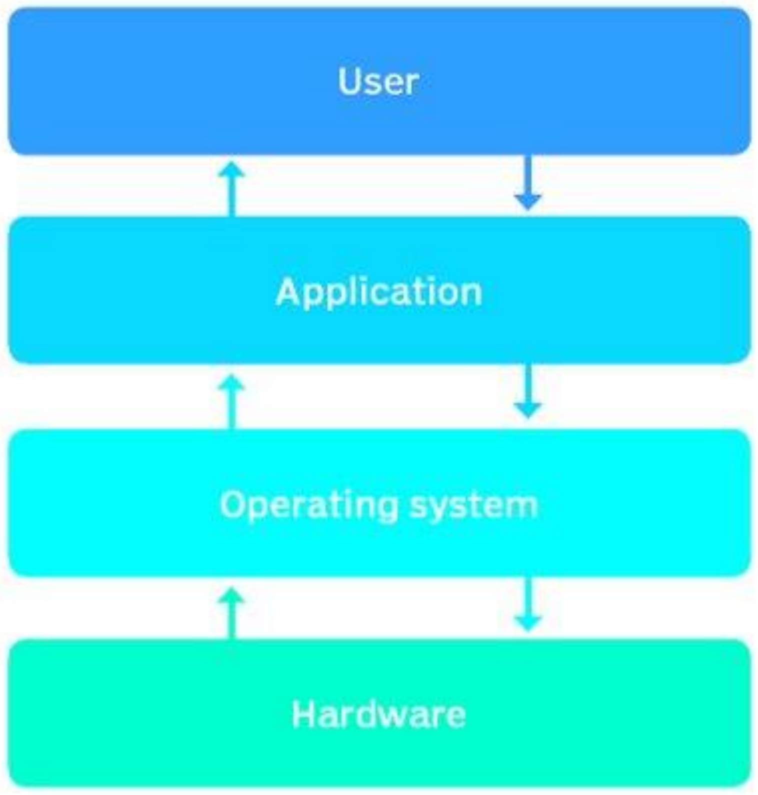
1) Cache Snapshot

2) Tracing process

3) Bus Monitoring

We will go into more detail about each attempt in the following chapters.

The following model depicts the different levels in operation on a standard computer.



The top level is the user level, the higher the level the more restricted the allowed operations are. On the user level, we are very limited in what hardware we may access, this level includes your basic I/O devices.

Next we have the application level, this level has less restrictions, in this level we can request memory allocation for example. This level is where most of programs we add upon the operating system exist.

Then we have the Operating System level. At this level we are theoretically non restricted and have access to all hardware and software running in the machine. There are some limitations, but only to keep things running in an orderly fashion.

The following configurations were used during the experiments:

- Ubuntu 20.04.2.0 (Focal Fossa) LTS
- Kernel 5.10.0

Our data sets run were:

- Normal use of a standard user (web, music, char, etc)
- Spec2006 benchmark (which turned out to be of little value, which will be explained later on)

Cache Snapshot

The first attempt we made was a naïve approach to simply read the cache in its entirety.

This is requires in addition to other things (which turned out to be impassable obstacles) that we take snapshots at a high enough frequency to ensure that no adjustments to the cache are made in between snapshots. Including page calling and evicting.

This is a gross interference in the systems natural function. Causing the entire system to slow down and the data retrieved may be random at best when predicting systems without snapshots being taken.

This by itself should have been enough to deter us from taking this route, but we chose to go ahead hoping to solve this issue in the future.

The next obstacle we encountered was, that taking snapshots, require memory. This very quickly became an issue. In addition to the fact that the snapshots were taking up memory; limiting the available memory for the other processes, it reached a point where most of the memory was full of recordings taken by the snapshot. Causing page faults to happen at a much higher frequency than would happen in neutral system.

Yet we continued on hoping that this too we can overcome.

Until we realized, that when we sampled the data, we were simultaneously overriding data and we could not be sure if we were collecting all of the data in a sterile fashion.

At this point we felt we reached an impasse and abandoned this attempt.

Tracing process

The next attempt we made was to trace individual processes. This way when a process causes a page miss, we can log the missing page along with the PID (process ID) and keep track of how the memory is being used by individual processes.

This attempt was to place a “wrapper” function around individual processes to store the required data at each page retrieval event.

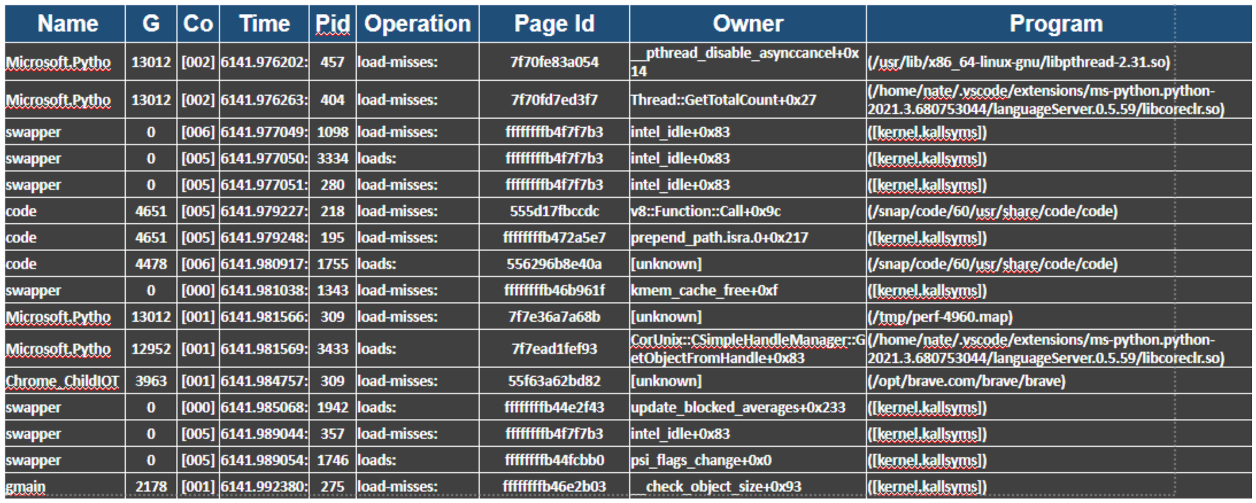
Very quickly we found that this was limited in it’s scope as, we could not wrap all processes running, this required an amount of work that could not be completed in a few months. In addition attempts to wrap kernel function was met with resistance as the kernel did not compile correctly and overall the experience was a failure early on.

Bus Monitoring (memory flux)

The final solution, which we found to have worked, was to listen to the flux of the cache. Meaning we take a given point of the cache (an initial state) and keep track of all ingoing and outgoing traffic. With this knowledge we will always know the state of the cache without have to look inside.

Our data extraction used the perf tool as a basis upon which we built our data exractor.

The following table displays the data we sampled:



Name	G	Co	Time	Pid	Operation	Page Id	Owner	Program
Microsoft.Pytho	13012	[002]	6141.976202:	457	load-misses:	7f70fe83a054	__pthread_disable_asynccancel+0x14	(/usr/lib/x86_64-linux-gnu/libpthread-2.31.so)
Microsoft.Pytho	13012	[002]	6141.976263:	404	load-misses:	7f70fd7ed3f7	Thread::GetTotalCount+0x27	(/home/nate/.vscode/extensions/ms-python.python-2021.3.680753044/languageServer.0.5.59/libcoreclr.so)
swapper	0	[006]	6141.977049:	1098	load-misses:	ffffffffb4f7f7b3	intel_idle+0x83	([kernel.kallsyms])
swapper	0	[005]	6141.977050:	3334	loads:	ffffffffb4f7f7b3	intel_idle+0x83	([kernel.kallsyms])
swapper	0	[005]	6141.977051:	280	load-misses:	ffffffffb4f7f7b3	intel_idle+0x83	([kernel.kallsyms])
code	4651	[005]	6141.979227:	218	load-misses:	555d17fbccdc	v8::Function::Call+0x9c	(/snap/code/60/usr/share/code/code)
code	4651	[005]	6141.979248:	195	load-misses:	ffffffffb472a5e7	prepend_path.isra.0+0x217	([kernel.kallsyms])
code	4478	[006]	6141.980917:	1755	loads:	556296b8e40a	[unknown]	(/snap/code/60/usr/share/code/code)
swapper	0	[000]	6141.981038:	1343	load-misses:	ffffffffb46b961f	kmem_cache_free+0xf	([kernel.kallsyms])
Microsoft.Pytho	13012	[001]	6141.981566:	309	load-misses:	7f7e36a7a68b	[unknown]	(/tmp/perf-4960.map)
Microsoft.Pytho	12952	[001]	6141.981569:	3433	loads:	7f7ead1fef93	CorUnix::CSimpleHandleManager::GetObjectFromHandle+0x83	(/home/nate/.vscode/extensions/ms-python.python-2021.3.680753044/languageServer.0.5.59/libcoreclr.so)
Chrome_ChildIOT	3963	[001]	6141.984757:	309	load-misses:	55f63a62bd82	[unknown]	(/opt/brave.com/brave/brave)
swapper	0	[000]	6141.985068:	1942	loads:	ffffffffb44e2f43	update_blocked_averages+0x233	([kernel.kallsyms])
swapper	0	[005]	6141.989044:	357	load-misses:	ffffffffb4f7f7b3	intel_idle+0x83	([kernel.kallsyms])
swapper	0	[005]	6141.989054:	1746	loads:	ffffffffb44fcbb0	psi_flags_change+0x0	([kernel.kallsyms])
gmain	2178	[001]	6141.992380:	275	load-misses:	ffffffffb46e2b03	__check_object_size+0x93	([kernel.kallsyms])

We ran this process across several different batches to get a lot of training material.

In the table we can see different processes running and the successful “loads” and the unsuccessful “misses”

The data we collected also gives us a group and PID which allowed us further analysis.

The time column was the most critical, as it gave us an indication of when the page was accessed, and if the page was accessed within the training timeframe, we can give a prediction in next time frame whether it will be accessed or not.

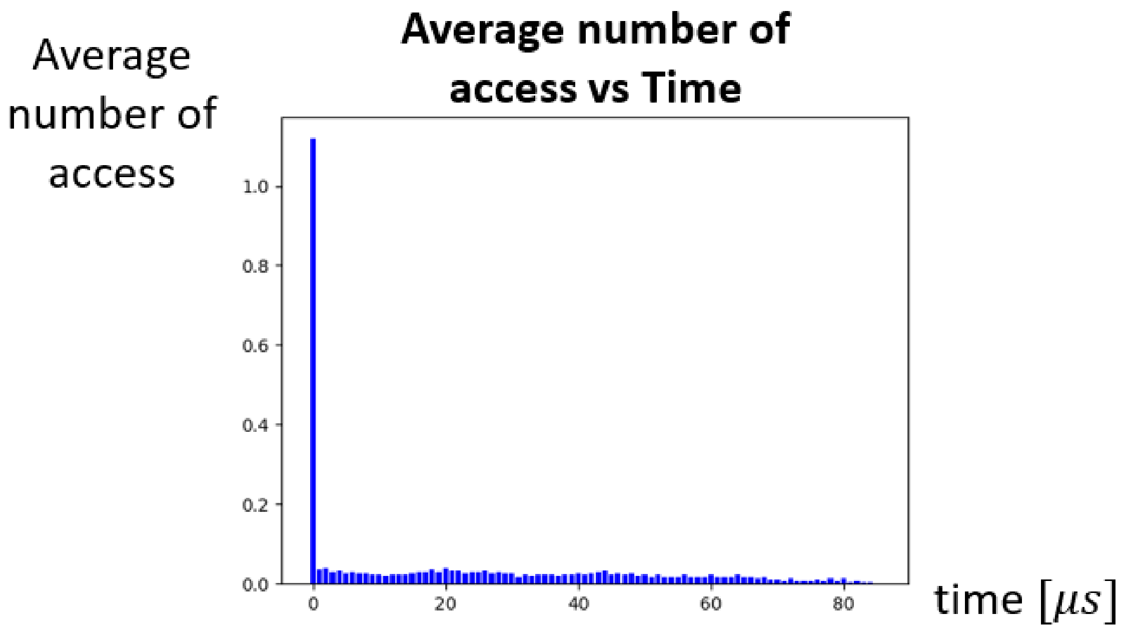
Data Analysis

Let us have a look at the data now.

Before we start feeding the data into the different models we wanted to have a look. Perhaps some form of pre processing to the data may be required. In addition data visualization may give us an insight into better ways to collect the data, how the system operates, things we may have missed, etc.

This is the first graph we would like to display.

Data Analysis



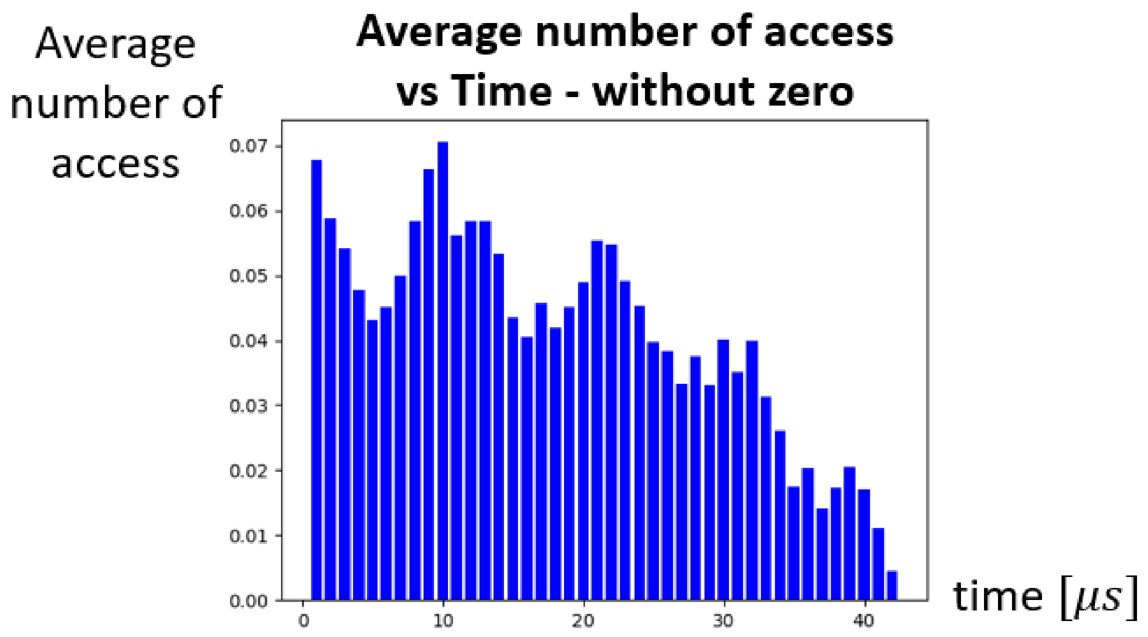
The graph is of Average number of times a page was accessed (average across all pages) in the different time intervals of its lifespan. When we say lifespan, we refer to the length of time it existed in the cache. Where the term birth refers to the moment a page was brought from remote storage to the cache, and death being the moment the page was evicted.

As would be anticipated the average number of accesses per page in its moment of birth is higher than one. This is to be expected as pages are only when born when a task requests access to it.

When can see from this graph that after the initial moment of access very few pages are accessed again within their lifespan.

This leads us to our next graph.

Data Analysis



This is the same graph as before only this time, the initial moment is removed to allow us to examine the smaller details of the graph.

As we can see, the graph has a downward trend. Meaning towards the end of a pages life, access to it becomes less frequent. This too is to be expected as the algorithm by which evict pages, evicts pages that have not been accessed lately.

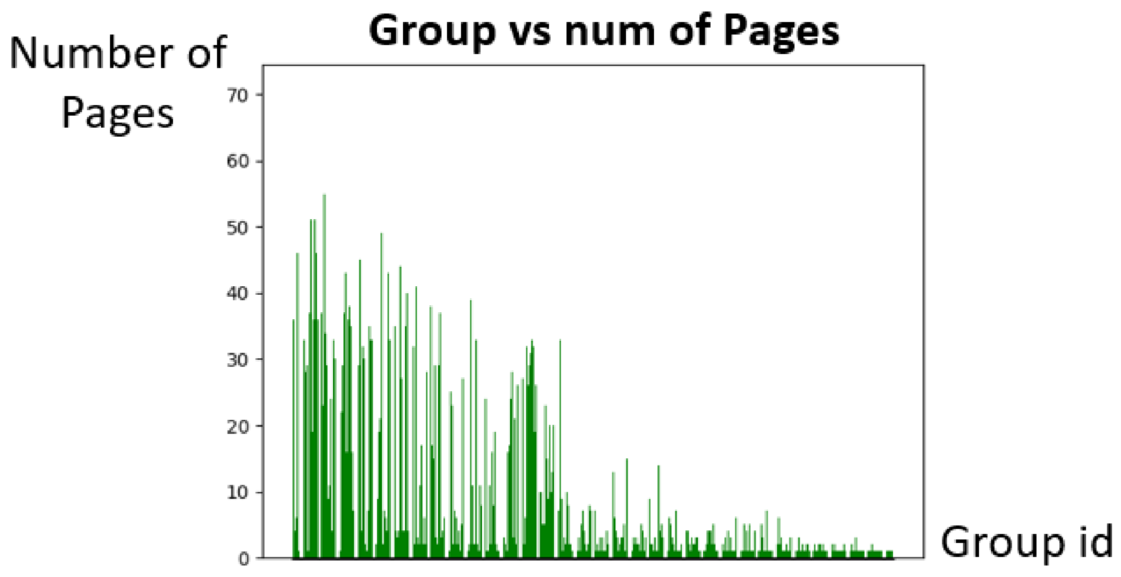
Another interesting thing to note, is the wave like function that seems to riding this downward trend.

The wave has a certain periodicity. This got us really excited, as this is an indication of the scheduling algorithm which grants time slices and here we can see the scheduler frequency.

This gave us confidence that our data extraction was successful as we sample frequently enough to match the schedulers frequency.

The next graph to analyze.

Data Analysis



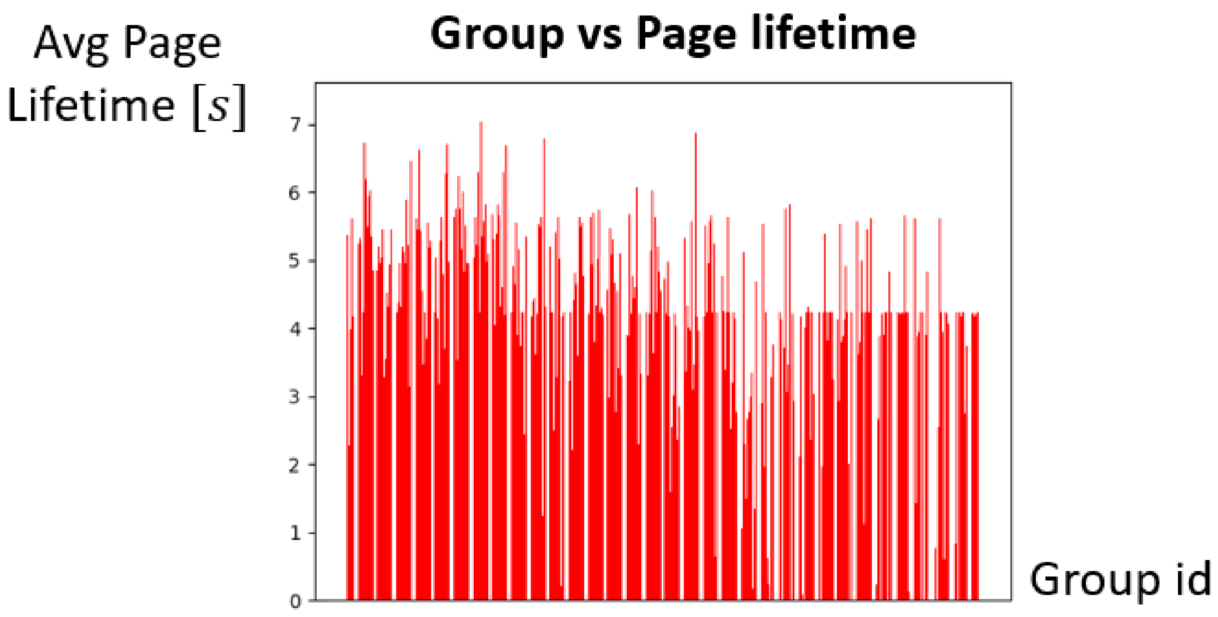
Number of pages per group in the cache. Where group is a defined by the operating system. There is a certain logic to the group numbering as is not totally random. Lower group ID's are usually given to kernel (operating system) tasks.

Knowing this we can understand the significance of the graph. Where it is clear lower IDs have more pages in the cache at a given time. This signals to us that the majority of pages in the cache are owned by operating system tasks.

This goes to show us the importance of collecting the data across the entire cache and why the second attempt at data extraction was not effective.

The final graph which we think is important to share is the following one.

Data Analysis

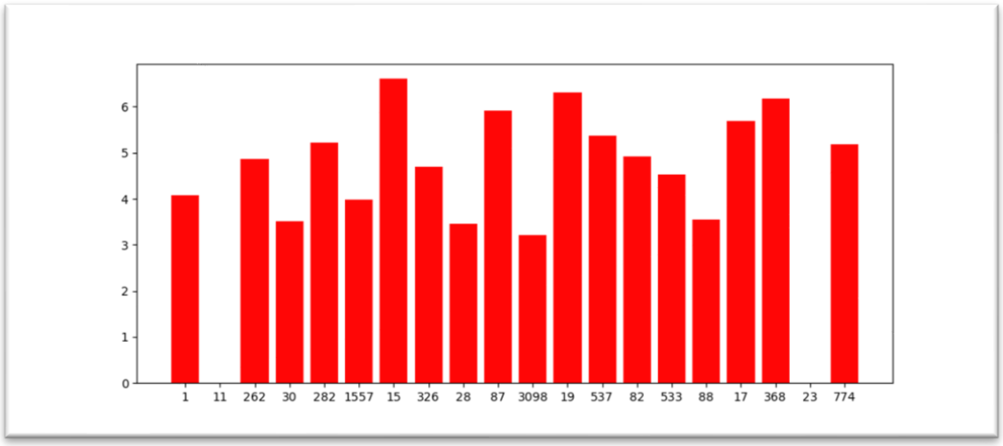


Page lifetime vs. Group ID. It is important to note that the lifetime is measured in seconds which in the computing world is a large unit. Some things to note, there is a “DC signal” what seems to be a constant “line” across a little above four seconds, around which the different lifetimes average.

With the higher end lifespans being nearly seven seconds long, a major discrepancy when compared with four seconds.

Another thing of interest is the fact that the lifespan of pages belonging to lower group IDs (those more likely to belong to the operating system) are longer.

For a closer up version, we displayed several random bars



Training models

The data we feed into our models, ended up being raw data we extracted, with the following features.

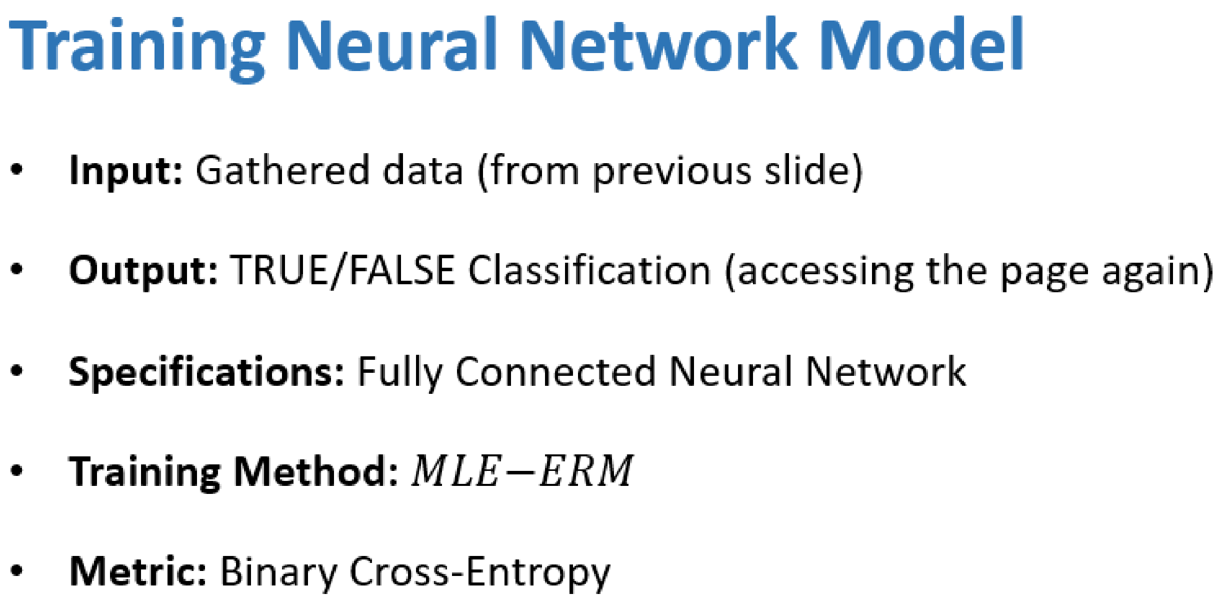
Process wakeup time, number of accesses to the memory of the process, number of R/W request to the page, the size of the group, the number of avg. W/R request of the group

Where our output is a prediction of whether a given page will be accessed in the next time interval. Where time interval is training data dependent, as we train on a given interval our predictions match that interval.

We trained in 20 seconds intervals, so our prediction is given a page with all the features mentioned above, will the page be accessed in the next 20 seconds.

The output is binary, TRUE/FALSE.

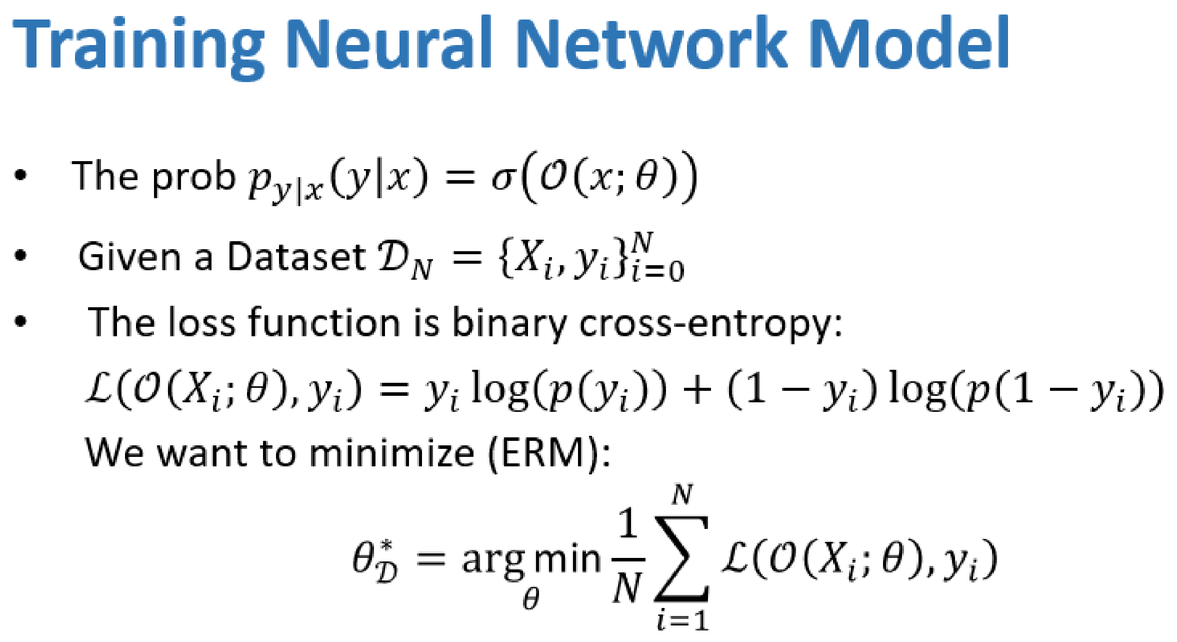
Here is a more complete specification of model:



Training Neural Network Model

- **Input:** Gathered data (from previous slide)
- **Output:** TRUE/FALSE Classification (accessing the page again)
- **Specifications:** Fully Connected Neural Network
- **Training Method:** $MLE-ERM$
- **Metric:** Binary Cross-Entropy

Some formality on how these are defined mathematically.



Training Neural Network Model

- The prob $p_{y|x}(y|x) = \sigma\big(\mathcal{O}(x;\theta)\big)$
- Given a Dataset $\mathcal{D}_N = \{X_i, y_i\}_{i=0}^{N}$
- The loss function is binary cross-entropy:

  $\mathcal{L}(\mathcal{O}(X_i;\theta), y_i) = y_i \log(p(y_i)) + (1 - y_i)\log(p(1 - y_i))$

  We want to minimize (ERM):

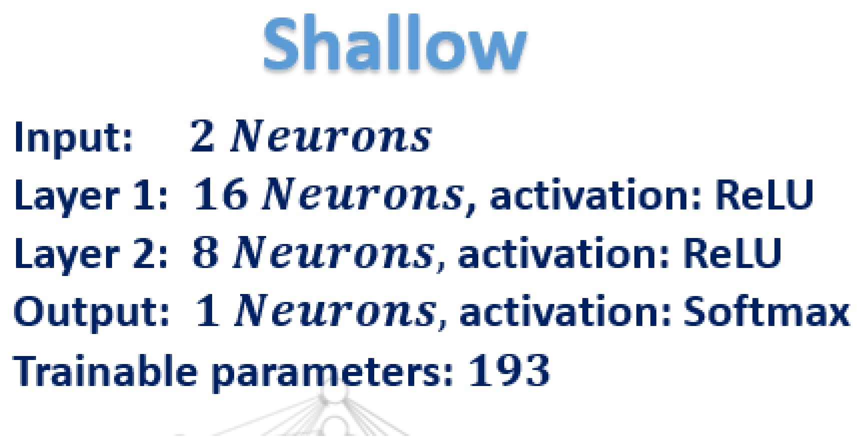
$$\theta_{\mathcal{D}}^{*} = \arg\min_{\theta} \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(\mathcal{O}(X_i;\theta), y_i)$$

We chose to work with two models.

One being a more shallow model, and one being a deeper one.

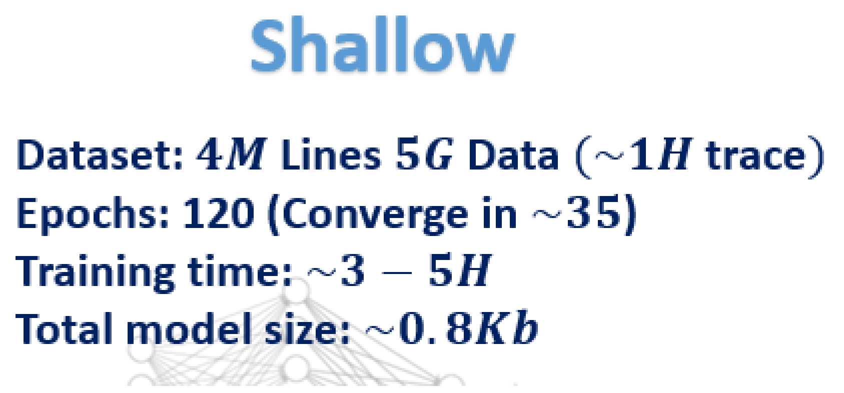
The thinking behind the two different models. Was that depending on the desired accuracy for prediction compared with time and power consumption can be decided by the user.

Shallow model



Shallow

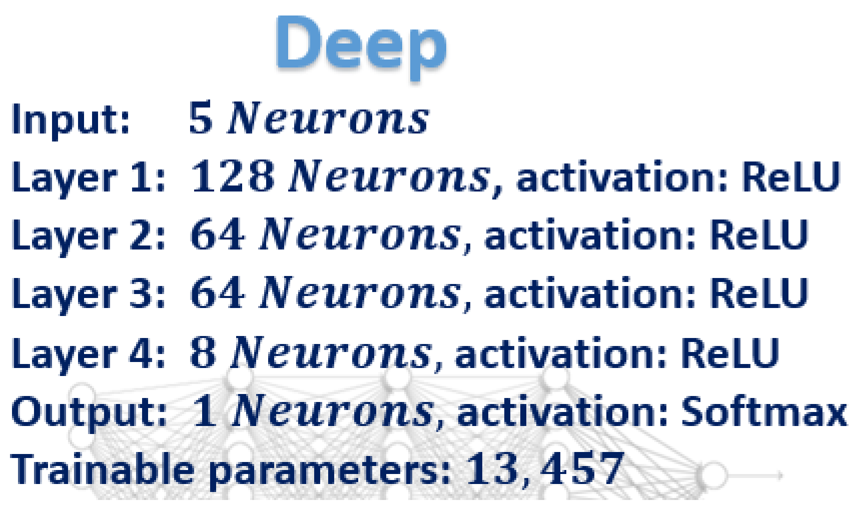
Input: $2\ Neurons$
Layer 1: $16\ Neurons$, activation: ReLU
Layer 2: $8\ Neurons$, activation: ReLU
Output: $1\ Neurons$, activation: Softmax
Trainable parameters: 193



Shallow

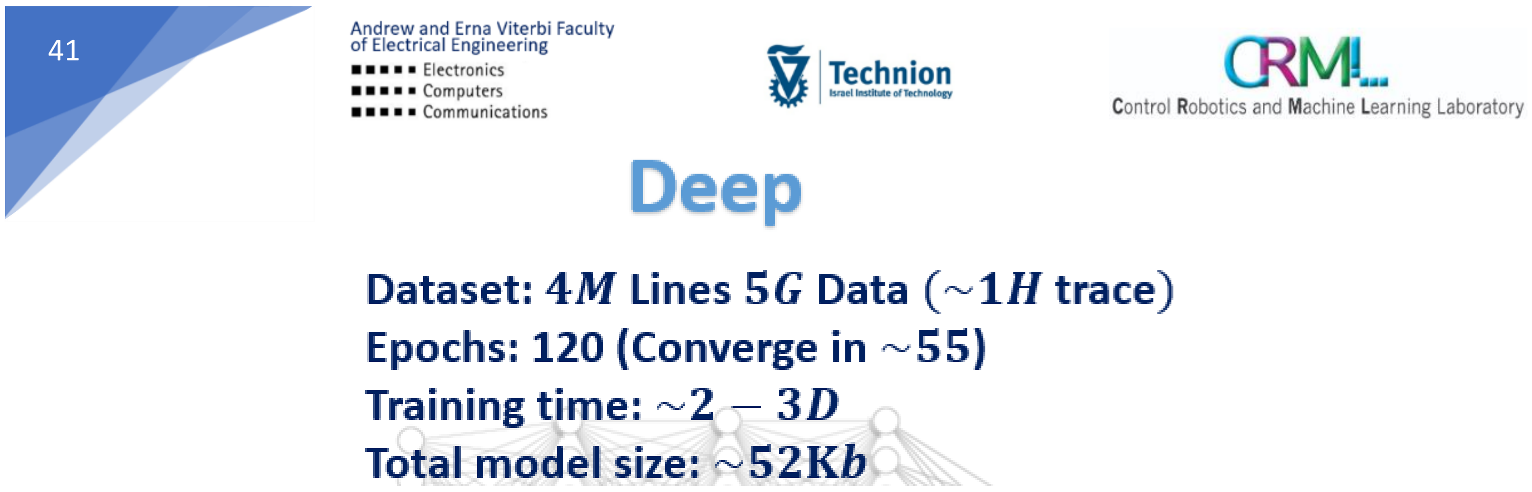
Dataset: $4M$ Lines $5G$ Data ($\sim 1H$ trace)
Epochs: 120 (Converge in ~ 35)
Training time: $\sim 3-5H$
Total model size: $\sim 0.8Kb$

Deep model



Deep

Input: $5\ Neurons$
Layer 1: $128\ Neurons$, activation: ReLU
Layer 2: $64\ Neurons$, activation: ReLU
Layer 3: $64\ Neurons$, activation: ReLU
Layer 4: $8\ Neurons$, activation: ReLU
Output: $1\ Neurons$, activation: Softmax
Trainable parameters: $13,457$



41

Andrew and Erna Viterbi Faculty of Electrical Engineering

Electronics Computers Communications

Technion Israel Institute of Technology

CRML Control Robotics and Machine Learning Laboratory

Deep

Dataset: $4M$ Lines $5G$ Data ($\sim 1H$ trace)

Epochs: 120 (Converge in ~ 55)

Training time: $\sim 2 - 3D$

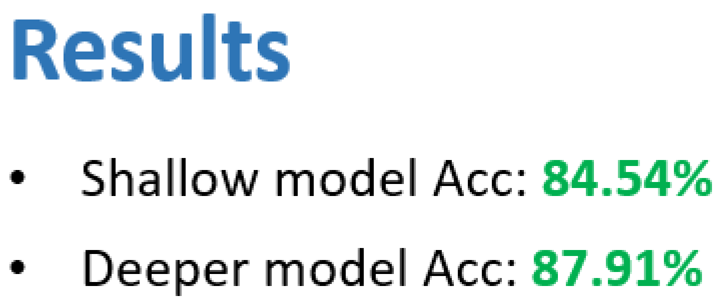
Total model size: $\sim 52\mathrm{K}b$

With the split between the data being a conventional:

60% Train, 20% Validation, 20% Test

Conclusions

Our results:



Results

- Shallow model Acc: **84.54%**
- Deeper model Acc: **87.91%**

We completed the project with mixed feelings.

On one hand we feel we made significant progress and overcame large obstacles.

Our predictions are better than a random “coin toss” and even outperform existing scheduling algorithm.

Yet on the other hand we feel that we are capable of higher predictions than the ones we ended up achieving.

Perhaps not all of the data we inserted in the training stage was necessary and we committed overfitting. Or perhaps changing the timeframes of our collection may lead to better results.

Even as we complete the project we feel there remain some open ended questions which would require more time to answer. Yet we feel satisfied with the results we did achieve,

[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