A survey of OS process scheduling in HPC

Charles Melis
Illinois Institute of Technology, Chicago, IL, USA
cmelis@hawk.iit.edu

***Abstract*—My report is a survey on Operating System process scheduling in High Performance Computing that aims to provide a general overview and a description of the hot research area. It will provide some research axis for future work. I gathered papers together and split them into 5 classes. These classes compares the papers with the same theme and goes into depth on how they manage to solve the specific problem they target. I will compare and link all the papers within a same category. I will propose areas to investigate for new research.**

***Keywords*—Process scheduling, Linux, General-Purpose, Performance, Energy Efficiency, Cyber-security, Multi-core oriented, schedulers, QoS, I/O.**

I. Introduction

The increasing evolution of performance in hardware devices such as High-Performance Computing requires a lot of study. We need to study the different scheduler algorithms that are created and implemented then they are compared to usual scheduler algorithms. The new kind of schedulers created doesn't only target performances but are classified into multiple categories.

High performance computing (HPC) is the ability to process data and perform complex calculations at high speed. To put it into perspective, a laptop or desktop with a 3 GHz processor can perform around 3 billion calculations per second. While that is much faster than any human can achieve, it pales in comparison to HPC solutions that can perform quadrillions of calculations per second. For decades the HPC system paradigm was the supercomputer, a purpose-built computer that embodies millions of processors or processor cores. Supercomputers are still with us; at this writing, the fastest supercomputer is the US-based Frontier , with a processing speed of 1.102 exaflops, or quintillion floating point operations per second (flops).

The OS process scheduling is the activity of a process manager that determines the removal of a process from the CPU. It also chooses another process to be executed by the CPU based on a strategy. The objectives of the process scheduler are to maximize the amount of task done per unit (throughput), to maximize the distribution of resources fairly (fairness). It is also to minimize the amount of time since a process is ready, and started to be executed (wait time), and to minimize the amount of time since a process is ready and it has been finished (response time).

The need to perform complex computations efficiently and effectively makes HPC rely on OS process schedulers. The process scheduler should find the best way allocate the CPU resources to processes without encountering any problem. This study will conduct a comparison between the existing schedulers, describe the problem they target and how they manage to solve it. The common everyday life processes will be presented in the next background section.

II. Background

The OS process schedulers are present and ubiquitous in every operating system. These scheduler algorithms are divided in two distinct categories. The first one is the non-Preemptive schedulers, they are used when a process terminates, or when a process switches from running state to waiting state. The second one is the Preemptive schedulers, they are used when a process switches from running state to ready state or from the waiting state to the ready state. These schedulers are presented in Figure 1.

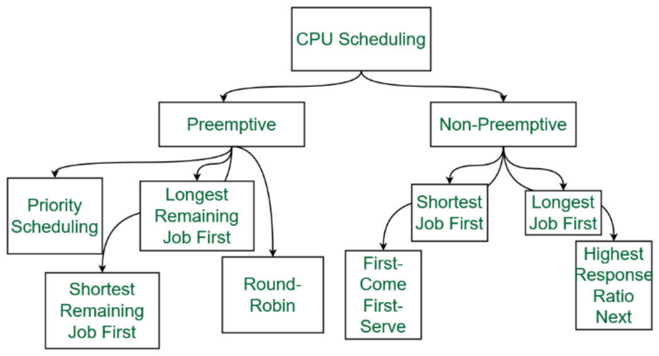


Figure 1: Different types of CPU Scheduling Algorithms.

I will first present the Non-Preemptive algorithm. Some of the schedulers above are really basic and common such as FCFS (First Come First Serve). First come first serve scheduling algorithm states that the process that requests the CPU first is allocated to the CPU first and is implemented by using FIFO (First Input First Output) queue. The Short Job First (SJF) is based on the lowest CPU burst time. The Longest Job First (LJF) is based on the highest CPU burst time. The Highest Response Ratio Next (HRRN) finds the response ratio of all available processes and select the one with the highest Response Ratio.

Among the Preemptive algorithms, a really famous one is the Round Robin algorithm, each process is cyclically assigned a fixed time slot (time quantum). It is the Preemptive version of the FCFS. Round Robin CPU Algorithm generally focuses on Time Sharing technique. With this technique all the processes get a balanced CPU allocation. Another algorithm is the priority scheduling that executes the highest priority task first. The Longest Remaining Job First and the Shortest Remaining Job First is respectively like the LJF and SJF but in a Preemptive way.

Most of the studies that I've read focuses on Linux schedulers. We must know that Linux OS has two types of processes. The Real-time processes that cannot be delayed and are used in process migration. There are also Conventional processes that are secondary and can be delayed if the system is busy. There are already some existing surveys in process scheduling. They will be described in the next section.

III. Related Work

In this section I'll explain the different work that has already been done. The first survey that I've read is "A Study of Available Process Scheduling Algorithms and Their Improved Substitutes" [1]. In this survey they compare the fundamentals process scheduling in Real-Time and Distributed systems. They compare the schedulers through:

1. Turnaround Time (TAT): It is the time taken for the completion of a job after its submission.
2. Waiting Time (WT): It is the time a process has to wait in ready queue before its execution.
3. Context Switch: In a preemptive multitasking system, a job is stored in memory to allocate CPU resources to another process. This stored job can then be restored at a further point in time so resume its execution.
4. Throughput: It is the number of jobs a system completes per unit time.
5. CPU Utilization: It is the amount of work the CPU is doing.

They compare only the common schedulers such as Round Robin, HRRN and FCFS the results are in Figure 2.

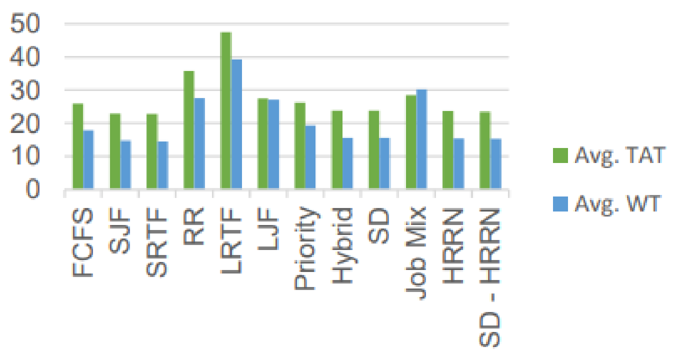


Figure 1: Performance of different Scheduling Algorithms.

The main drawback of this study is that they study the performance of common process schedulers and not the performance of new process scheduling and their main aspects.

Another survey that I've read that targets OS process scheduler is the one conducted by Jha and al. [2]. This survey is also a study of common CPU scheduling algorithms but focuses more on the hybrid CPU scheduling algorithm that eradicates the drawbacks of the Round Robin algorithm. The survey [3] propose a classification of different scheduling techniques that are used and compare their performance in term of CPU utilization, Throughput, Average Turn-around time, Average Waiting Time, Response Time, Fairness and Context Switching. The classification has been made by types of algorithms and not by the problem they aim to solve.

Another survey that I have read focuses on the different machine learning approaches to improve process scheduling [4]. The authors review a number of different machine learning approaches that have been proposed for process scheduling, including decision trees, neural networks, genetic algorithms, and reinforcement learning. They discuss the advantages and disadvantages of each approach and provide examples of how each approach has been applied in practice.

My survey has a different approach from all of the above. I gathered the papers that I've read by categories of the problem they solve. I'll explain their general functioning, their results and compare all the papers of the category to each other. Then in another section, I'll discuss them. The categories found will be described in the next section.

IV. Process schedulers technologies

In this section, I will explain the categories that I have identified. I will explain each paper related to them and compare them together. The categories that we found so far are : General purposes, Performance oriented, Energy efficiency, Cybersecurity oriented and multicore oriented. I'll first describe the General purpose oriented.

A. General purpose

In this subsection, I'll treat the papers that doesn't have only a goal of performance but they also have the goal to improve latency, scalability and load balancing. These papers can also aim to create a system less complex and more flexible.

The paper "A new intuitionistic fuzzy rule-based decision-making system for an operating system process scheduler" [5], aims to provide an algorithm that take into account the uncertainty and ambiguity in real-world systems. This scheduler takes into account the CPU utilization, memory utilization, and process priority. The algorithm uses a bunch of fuzzy rules to make a decision on which process should be executed first. They say that their algorithm can be used in real-world such as cloud computing or embedded systems.

Another paper that impacts on flexibility and performance is the ghOSt algorithm [6]. The ghOSt OS process schedulers, provides a delegation of scheduling processes to user space processes in a Linux environment. The new changes of ghOSt is that it decouples the policy definition from the kernel scheduling. In ghOSt, the normal Linux processes (agents) interact with the kernel thanks to the ghOSt API. You can see a schema of the scheduler in Figure 2.

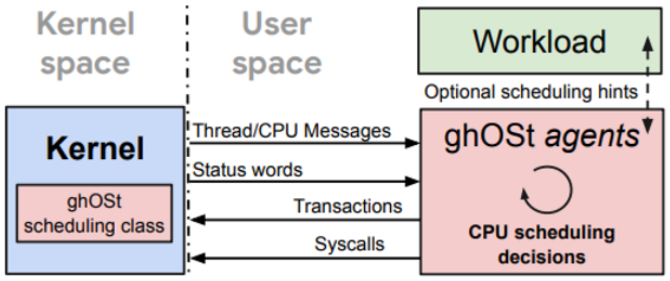


Figure 2: Overview of ghOSt.

The kernel notify the agents state changes for all the thread thanks to message queues in an asynchronous path. Then the agents use the API to commit scheduling decisions for the threads. The scheduling policy logic is implemented in the user space agents. The agents can be written in any language and debugged by standard tools, making them easier to implement and test. To achieve fault tolerance and isolation, if one or several of the agents crash, the system will fall back to the default scheduler, such as CFS. In case of bugs they have to relaunch the use space agent. The results show that GhOSt can provide performance improvement in addition to flexibility and an easy deployment.

To compare the ghOSt paper described above with a paper related to it, I'll talk about Plugsched [7]. Plugsched compared to ghOSt is safe and easy to use. It dynamically updates the kernel scheduler without any constraints. This tool created in 2023 use modularization: Plugsched decouples the scheduler from the Linux kernel to be an independent module. Then it uses the data rebuild technique to migrate the state from the old scheduler to the new one. Contrarily to the ghOSt scheduler, Plugsched doesn't require the two context switches between the user-level to the kernel level. Furthermore, Plugsched doesn't require a reboot of the operating system because it doesn't modify the kernel code. In their paper they provide a figure that compare ghOSt with Plugsched. You can see this Figure in Figure 3.

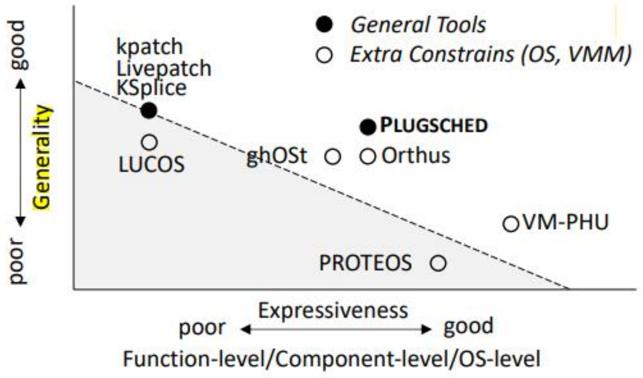


Figure 3: Comparisons between potential solutions

We can see that the Plugsched has a better "Generality" and a better "Expressiveness". The Expressiveness is the scope of a variety of optimization targets that developers can express. The Generality is how easy new features can be practical to Linux without extra constrains.

The two latter papers aim to provide flexibility to the schedulers, but they doesn't consider I/O. That is why I read the paper named "Horae: A Hybrid I/O Request Scheduling Technique for Near-Data Processing-Based SSD" [8] was created in 2022. This paper is designed for near-data processing-based solid-state drives (SSDs). Near-data processing (NDP). It involves moving computation closer to the data, reducing data movement and communication overheads. The proposed Horae technique combines two scheduling methods: time-based scheduling and priority-based scheduling. Time-based scheduling involves scheduling I/O requests based on their arrival time, while priority-based scheduling assigns different priorities to I/O requests based on their importance. By combining these two techniques, Horae can achieve high performance while also ensuring fairness and preventing starvation. The experiment of their tool shows that they outperform most of the existing schedulers.

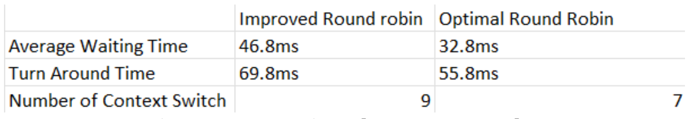
After presenting the paper that have general purposes such as ghOSt or Plugsched, I'll describe paper that are based on the improvement in the next section.

B. Performance oriented

After having described the general purpose papers that generally aim to target flexibility and usability problems in real life environments, we will discuss about the papers that propose studies that are specifically made to improve the performance of a scheduler or compare existing performances.

The first 1 of this category is the "An Improved Round Robin Algorithm for an Efficient CPU Scheduling" [9]. The goal of this paper is to counter the problems that a normal round robin algorithm could have such as overhead caused by context switching and the potential for starvation of processes with higher CPU burst times. To solve this problem they propose to dynamically adjust the time quantum assigned to each process based on its CPU burst time and the number of times it has been preempted. They showed that their tool has better performance than normal round robin algorithm in terms of Average Waiting Time, Average Turn Around Time and Context Switches.

To compare the paper [9] created in 2022, I'll talk about "Optimized Round Robin CPU Scheduling Algorithm" [10]. In this paper they also try to improve the existing round robin by adjusting the time quantum as well. The optimized time quantum value is evaluated based on many different features which are obtained as the set of features. To extract features, three steps are followed, namely, document organization, calculating list of number of cycles for each process, and setting parameters for feature set extraction. The feature extraction phase is then followed by building a new multiple linear regression analysis based model for identifying the optimized value of Time Quantum (TQ). This approach provide a better performance than the improved round robin algorithm. You can see the comparison in Figure 4.



	Improved Round robin	Optimal Round Robin
Average Waiting Time	46.8ms	32.8ms
Turn Around Time	69.8ms	55.8ms
Number of Context Switch	9	7

Figure 4: Comparisons between IRR and ORR

We can see thanks to the table above that the overall performances of ORR outperforms IRR. ORR has less context switches and better Average Waiting Time and Average Turn Around Time.

The two papers described above propose a new algorithm to improve the round robin. But some other papers just aimed to realize a benchmark between the Linx CFS (complete Fair Sharing) and the windows 10 CPU scheduling [11]. The results show that Linux CFS provides better interactivity performance than the Windows 10 CPU scheduler. The authors attribute this to the more aggressive scheduling policy of the Linux CFS, which prioritizes interactive tasks over CPU-bound tasks. The study also shows that the perceived responsiveness of interactive tasks is closely related to their response time, with shorter response times resulting in a better user experience.

The papers cited above doesn't take into account the real-time event processing. That is why we introduce the paper "Improving the performance of Real-Time Event Processing based on Preemptive Scheduler FPGA Implementation" [12]. They first insist on the fact that traditional algorithms doesn't have the optimum performances due to the overhead of the OS and the CPU. They propose a Preemptive scheduler based on Field-Programmable Gate Arrays (FPGAs). Field-programmable gate arrays (FPGAs) are reprogrammable integrated circuits that contain an array of programmable logic blocks. Their implementation is designed to handle real-time event processing tasks with low latency and high throughput. They show that their solution outperform general software based algorithms.

Creating and implementing some new mechanisms to improve the performance of process scheduler is a huge part of the researches now. But these algorithms needs to consume energy with efficiency. I will talk about the energy efficiency papers that I've read that aims to solve this problem in the next section.

C. *Energy Efficiency oriented*

The main goal of this category is to describe papers that propose new technologies to promote reliability, cost-effectiveness and extended battery life. They aim to save the energy consumption and provide a balance between the execution costs and execution time.

The 1st paper that I'll present in this category is "Compatibility of Hybrid Process Scheduler in Green IT Cloud Computing Environment" [13]. In this paper they first discuss how important it is to have an efficient algorithm. Then they propose the CHPS: Compatibility of Hybrid processor scheduler. CHPS decides which resources should be chartered from the public cloud and combined to the private cloud to offer adequate processing power to perform a workflow inside a specified execution time. CHPS has 3 phase to work : The first phase describes the process of identifying the pool of processors and assignment of those processor based on time and task schedules. The second phase described the process of allocating the resources to a set of processors in an optimal manner. The third phase describes the Compatibility of Hybrid Process Scheduler in Cloud Computing with old, mid-range and modern processors suiting to various operating system environments and application needs. They showed that their algorithm outperforms the HCOC [14] : The Hybrid Cloud Optimized Cost scheduling algorithm. HCOC is an algorithm to dominate the implementation of workflows following a preferred execution time, but also reducing the costs when contrast to the resource consuming approach. Here is the comparison that CHPS made in figure 5.

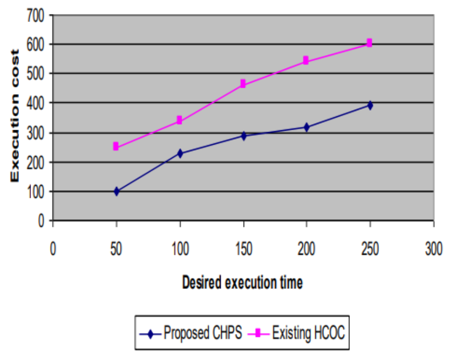


Figure 5: Execution Time vs Execution Cost

The two papers above propose a new algorithm to reduce the cost of execution in hybrid cloud.

The paper [15] proposed an algorithm to be energy aware in heterogeneous multicore systems. They first bounded the shared cache unpredictability by adopting cache partitioning techniques where specific sections of the shared cache are individually assigned to cores to prevent concurrent tasks from accessing the same cache lines. Secondly, they probe the impact of core-frequency and cache-partitions on task execution-cycles and its resultant impact on the execution time and energy consumption. This paper is closely related to two above because in hybrid computing, you might encounter different cores types with varying architectural and micro-architectural features. This diverse computing potential is desirable since tasks may differ in their affinity to different core-types. The result of this paper shows that the proposed algorithm achieves significant energy savings (compared to the generic algorithms) while meeting the performance constraints.

As we saw the importance of the energy consumption in process scheduling, we also saw the impact of schedulers in multicore environments. I'll speak about papers treating the multicore environment in the next section.

D. Multi-core system oriented

The papers of this section aim to solve the multi-cores general problems. The paper [15] described in the last section could also fit in this category because they take into account the variety of existing architectures, and they explain their algorithm to say which core should execute which process.

To first understand the main existing algorithm on multi-core scheduling, I read a paper named "Survey of scheduling techniques for addressing shared resources in multicore processors" [16]. This survey provide an overview of schedulers such as the Building Blocks of a Contention-Aware Scheduler, this type of scheduler takes into account the contention for shared resources. They include resource monitoring, performance models, scheduling policies, workload characterization, resource allocation mechanism. They also present a some complementary techniques such as DRAM Controller Scheduling, or Cache Partitioning. DRAM controller scheduling involves scheduling requests to access the main memory (DRAM) in a way that reduces contention for the memory bus and improves overall system performance. Cache partitioning involves dividing the shared cache among the cores in a way that reduces contention for the cache and improves overall system performance. Then they discuss the Solaris and Linux schedulers.

Once having understand the key concepts of multicore schedulers, I read the paper [17], it focuses on multi-core systems but for mobile operating systems. They says that the power of the core of the mobile is not exploited as its best so they provide an extended scheduler that assigns a process to a particular core to reduce cache misses, they adjust the frequency of the processor based on the workload of the running process, they balance the workload between the core to ensure that one core is not overloaded, they allow user to assign the priority that they want for a software. The main problem is that this paper is only for mobile OS.

With the issue of the last paper I've decided to read [18], which propose a way to have an efficient process scheduling in multi-core systems. They propose the Longest Path First In (LPFI) algorithm. LPFI analyses the dependency relationship between all processes and generate a dictionary whose key is different processes name and whose value is the maximum distance between the key and other processes that have direct dependency relationship with the key. Then, LPFI will allocate different processes one by one according to their value within the dictionary in decreasing order so as to give priority for processes that have dependency longer chain. The result shows that their method is 10% more efficient than Integer Linear Programming (ILP) that encodes all processes dependencies and the limit of processes number at one time point into parameters and constraints into this scheduling problem and leverage the developed ILP solver to finish the scheduling problem automatically.

The paper [19] is an algorithm that makes intelligent decision on asymmetric multicore processor in order to have the exploit the full potential. They propose a COLAB system. The system is divided into a runtime scheduling process and an offline modelling processes. The runtime scheduler is built inside OS kernel and composed of i) a core allocator to handle fairness and core sensitivity; ii) a thread selector to achieve bottleneck acceleration; and, iii) machine learning based runtime models, which predicate speedup and power consumption of threads on heterogeneous cores. The offline modelling processing is applied to construct runtime models by offline training. They show that their algorithm outperform Linux CFS by 25% in turnaround time and average throughput.

A paper that is also relied to the ones above is [20] "A Novel Hard-Soft Processor Affinity Scheduling for Multicore Architecture using Multiagent". The paper proposes a novel approach to processor affinity scheduling for multicore architectures using a multiagent system. The multiagent system consists of several agents, including a task agent, a processor agent, and a coordinator agent. The task agent represents the task to be executed and provides information about its hard and soft processor affinities. The processor agent represents the core and provides information about its processing capabilities. The coordinator agent manages the scheduling process and selects the appropriate core for each task based on its hard and soft processor affinities. As you can see in Figure 6.

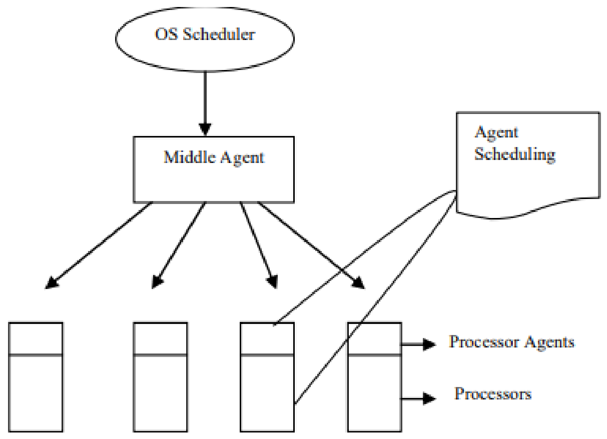


Figure 6: Process Scheduling by OS scheduler, middle agent and individual agents

They show that they have a better CPU utilization when the number of processes grow compare to Round Robin or Shortest Job First as you can see in figure 7.

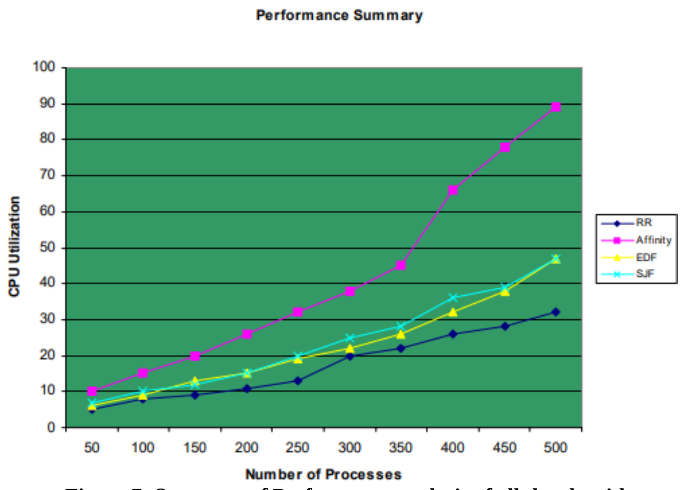


Figure 7: Summary of Performance analysis of all the algorithms

We saw that the schedulers are created to target different problems such as the energy efficiency, make the system more flexible, improve the performances and adapt to the multi-core architecture but they also target some cybersecurity issues. I'll present these schedulers in the next section.

E. Cybersecurity oriented

A paper that represents well this category is the [21], they show that real-time system such as modern automobile, avionics systems and UAVs (unmanned aerial vehicles) can be vulnerable to malicious code injection. They assume that an attacker can insert new tasks or compromise one or more existing tasks. This paper aim to reduce this potential for information leakage via shared resources by integrating security at the design phase of RTS in the form of intelligent scheduling constraints. They discuss various methods of integrating such constraints into scheduling policies for real-time systems. They focus on Fixed priorities scheduling because it covers a large amount of real-time system. The information of a high priority task must not leak the information of a low priority task. Since the shared resource is the offending party, we can ensure that it is always flushed/cleaned out whenever we see transitions of the type High priority → Low priority; Ensure that all processes of one kind complete before transitioning to the other; or prevent Low priority processes from being preempted by High priority processes once it has been scheduled. This is how they managed to secure the real-time systems.

The paper [22], fits into this category but could also fit in the energy category. The proposed scheduler, called SENAS, uses a combination of energy-awareness and security-driven scheduling to improve system performance and reduce energy consumption while ensuring security for real-time approximate computing tasks. They first provide an analysis of various threats for a real time approximate computing application, depicted as a PTG (Precedence-constrained Task Graphs). They present the development of a scheduling strategy that schedules AC (approximate computation) tasks and determines the amount of optional portions that need to execute based on availability of energy budget. They present the development of a self-aware runtime security mechanism that detects anomaly in processors at runtime and takes appropriate measures to ensure completion before deadline. They create a low overhead SENAS (Security driven ENergy Aware Scheduler) that performs the two latter development , which maximizes QoS in the available energy budget. The SENAS has 2 modules. The scheduling module is associated with mapping appropriate task versions to the processors, based on available energy budget at that time. The security module continuously monitors the working of the processors at runtime and on detecting any anomaly, calls the scheduler module to develop a fresh schedule based on available energy at that time. They showed that the QoS and the success rate are related with the energy factor. the ideal scenario when additional energy budget is almost 75%.

To conclude this section, I'll present the paper "Migrating an OS Scheduler into Tightly Coupled FPGA Logic to Increase Attacker Workload" [23]. The paper propose a way to increase the workload of attackers by migrating an OS into a FPGA logic. The authors propose a hardware/software co-design approach to implement the scheduler in FPGA logic, which enables faster context switching and reduces the overhead of software-based scheduling. They increase the workload by introducing additional complexity and reducing the amount of available resources for the attacker. They hide core OS component into FPGA logic. A rudimentary round-robin scheduler is first implemented in software on the PS. Then, baseline performance and security metrics are obtained. Next, the scheduler is implemented in the FPGA logic, and performance costs and security enhancements are characterized. All trials use a dynamic depth process ready queue. The software implementation uses a queue structure with dynamic memory allocation. The hardware implementation uses an equivalent block-RAM based hardware FIFO for the process ready queue. These findings demonstrate that migrating an operating system's scheduler control logic and data structures into tightly-coupled FPGA logic can both increase attacker workload and improve overall system performance. The best process scheduler code-size reduction (~90%) and performance speedup (~50%) were achieved when the conditions were optimal. When the conditions were not optimal they show an appreciable speed-up (~10%) and a significant code size reduction (~80%). The counter part is for high level approach, they had a huge performance cost of (~55%). The approach of this paper is interesting because it is different from the two other paper but still aim to solve a cybersecurity problem.

After having reviewed the 5 categories mentioned earlier, and described the papers that fits into these categories, I'll discuss these papers in the next part.

V. Discussion

In this section we will discuss the paper of the main categories. We will provide thoughts and limitations category by category.

First in the general purpose category, we saw that the papers aims to provide a better flexibility such as the ghOSt operating system. The main drawback of this paper is that it has context switches to communicate from user-level space to kernel level space plus it needs to reboot the OS because it modifies the kernel. That's why they created the Plugsched paper to provide an update without the need to reboot the OS because the mechanism is at the user-level space. The Plugsched algorithm has been deployed and proved to be efficient in real-world cloud. The tests has been realise on 4000 servers on an internal cloud. Future research can test the Plugsched algorithm and provide solutions to adapt it in external/ hybrid cloud. Concerning the fuzzy analyser, the result in terms of performance are the same as already existing algorithms but they can be improved by identifying the sleep time spent either by a batch or a process.

The second category we presented typically performances oriented. The first two papers discuss the time quantum of a slice that is injected into a CPU to be processed. Both proposed interesting algorithms. The ORR has shown better results than the IRR. The problem with this algorithm is that the experience is only experimental and has not been tested in real-world environment and the algorithm for the ORR is based on a dataset. So depending on the dataset the results might change plus the dataset is not proven to be adapted to an industrial use. In this category, I also described a benchmark between the Linux CFS and the windows and I explain that the general Linux CFS outperforms the windows scheduler. The main drawback of this benchmark is that it has been test with Audio/video programs and not programs that require a lot of processing like the supercomputers. In the performance section, I also described a preemptive scheduler for real-time operating systems. This scheduler can be enhanced by designing a debug and trace module to enable better control of the system and a high diagnostic capability, providing system designers with flexible responses to the imposed requirements.

In the third section, I described energy oriented schedulers. The two first schedulers such as CHPS and HCOC are all based for hybrid cloud applications. The CHPS paper take into account the resource usage such as bandwidth, CPU cycles, I/O operation, Energy, and data transfer rate, but doesn't provide any satisfaction on the QoS. The HCOC doesn't take into account the general resource usage of the machines. The third paper described is a little bit related to the two other because in a cloud you can find heterogeneous multicore but it is not focused on cloud utilisation. They show an average and maximum energy savings of 14.9 and 20.4 percent, respectively for core-level energy consumption, and 20.2 and 60.4 percent, respectively for system-level energy consumption. But this paper doesn't provide a specified enough results that their algorithm had in the core performance. A future search could be to quantify the study of this typical algorithm on performance depending on the core type.

In the fourth section, I'll discuss the multi-core scheduler that I presented. The goal of these paper is to permit a program to use the full capacity of the cores of the machines. One paper presented the LPFI that record the longest chain to input into the CPU. This paper try to maximum balance the workload between the cores on the same machine but doesn't provide any thoughts on how can it be applied to distributed systems with different cores. They take into account the different previous improvements to enhance the QoS. COLAB demonstrates the applicability in realistic scenarios, allowing better execution times and energy efficiency for parallel workloads on AMP processors without additional effort from the programmer. The COLAB paper system is based on a machine learning that is trained before, so the result depends on the training. But their experiment shows a good adaptation on real world. The paper that propose a multiagent technology is limited by the hardware in terms of : high off-chip memory bandwidth, the high cost to migrate a process, the small aggregate size of on-chip memory, and the limited ability of the software (agents) to control hardware caches. The scheduler can also be extended for distributed systems that can deploy multicore environment. We can also implement an efficient load sharing with the help of multiagents as they cooperate with each other and get the status information of every other processor in the multicore chip

In the fifth section, we discussed cybersecurity topics. Cybersecurity should be considered for schedulers in order to no leak any information through the processes and to quantify the impact of a cyberattack on the schedulers in term of performance and risks possible. We presented one paper that respect the confidentiality of the CIA triad by preventing some data leakage. The second algorithm tries to anormal event on a scheduler to see when an attack takes place. The 3rd paper has been created to reduce the attacker power during an attack and can be really use full on real world problems. For example a ransomware uses a lot of power to encrypt/ spread. Reducing this power can help the security team to do mitigation while an attack is propagating. Most schedulers must incorporate a security part because cybersecurity become more and more prevalent as the technology grows.

In this paragraph, I'll discuss what is lacking in the existing schedulers. Some of the schedulers that I read doesn't consider the I/O characteristics nor network characteristics. These schedulers should be tested with a more constraints. Some other paper do consider them and organize their tool depending on that. Most of the scheduler are tested in lab environments but never tested in production. As we know that labs and real environment are not the same, after having demonstrated their efficiency in a laboratory should be tested in the real world. Some paper consider QoS and can provide QoS, due to their average results during the testing part. In general an application can communicate to the scheduler that it is waiting for a thread to be executed thanks to synchronization primitives. But from the papers that I've read, most of them doesn't take into account the status of an application to enhanced their performances. I think that future work should take into account the applications status. Some schedulers take into account the thread dependencies/chain but not when an application tells the scheduler that it is waiting for a thread to be executed. Each schedulers can be improved as I said in the sections above. I my opinion new schedulers should combine the categories above. A new interesting scheduler might be able to fit in all the sections described. A "perfect" scheduler might be able to provide high flexibility by being implemented without the need to reboot the OS. This scheduler should consider to improve performance and compare with the latest scheduler such as plugsched. Meanwhile improving the performance the scheduler will need to provide energy efficiency without degrading the performances obtained. This performance can be enhanced by utilizing the full power of a core. But in the overall, the scheduler has to be secure by disabling data leakage or data injection.

During my survey, I tried to take a look at recent papers (between 2020 and 2023), only 5 papers are before 2020. In the papers after 2020 that I've read, none of them can be included in all the sections that I've cited.

VI. Conclusion

In this survey, I read 23 papers. The first ones where surveys in order to see what work on the same topic has already been done before. The other papers aim to solve various problems using various approaches. The main problems that they aim to solve are the flexibility and usability, the performance improvement, the energy efficiency mostly in a cloud usage. Another category was to improve the utilization of the core in a multicore environment. The last goal was to target cybersecurity problem, either by removing the data leakage vulnerability or preventing an attacker to use the full power of the computing resources. Most of the paper in these categories provide results from lab environment but not on real-time applications. These paper propose novel technics to implement new ways to schedule while solving hot research area topics.

References

[1] Kulkarni, Nahush & Mahajan, Sameer & Patil, Adwait. (2020). IRJET | A Study of Available Process Scheduling Algorithms and Their Improved Substitutes. 7. 4129-4136.

[2] Chowdhury, Subrata. (2018). Survey on various Scheduling Algorithms. "Imperial Journal of Interdisciplinary Research (IJIR). 3. 4.

[3] Harki, Naji et al. "CPU Scheduling Techniques: A Review on Novel Approaches Strategy and Performance Assessment." (2020).

[4] Dias, Siddharth & Naik, Sidharth & Kotaguddam, Sreepraneeth & Raman, Sumedha & M., Namratha. (2017). A Machine Learning Approach for Improving Process Scheduling: A Survey. International Journal of Computer Trends and Technology. 43. 1-4. 10.14445/22312803/IJCTT-V43P101.

[5] Butt, Muhammad Arif & Akram, Muhammad. (2016). A new intuitionistic fuzzy rule-based decision-making system for an operating system process scheduler. SpringerPlus. 5. 10.1186/s40064-016-3216-z.

[6] Jack Tigar Humphries, Neel Natu, Ashwin Chaugule, Ofir Weisse, Barret Rhoden, Josh Don, Luigi Rizzo, Oleg Rombakh, Paul Turner, and Christos Kozyrakis. 2021. GhOSt: Fast & Flexible User-Space Delegation of Linux Scheduling. In Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21). Association for Computing Machinery, New York, NY, USA, 588–604. https://doi.org/10.1145/3477132.3483542

[7] Teng Ma, Shanpei Chen, Yihao Wu, Erwei Deng, Zhuo Song, Quan Chen, and Minyi Guo. 2023. Efficient Scheduler Live Update for Linux Kernel with Modularization. In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3 (ASPLOS 2023). Association for Computing Machinery, New York, NY, USA, 194–207. https://doi.org/10.1145/3582016.3582054

[8] J. Li et al., "Horae: A Hybrid I/O Request Scheduling Technique for Near-Data Processing-Based SSD," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 41, no. 11, pp. 3803-3813, Nov. 2022, doi: 10.1109/TCAD.2022.3197518.

[9] A. M. Oshani Bandara and U. U. Samantha Rajapaksha, "An Improved Round Robin Algorithm for an Efficient CPU Scheduling," *2022 2nd International Conference on Advanced Research in Computing (ICARC)*, Belihuloya, Sri Lanka, 2022, pp. 349-354, doi: 10.1109/ICARC54489.2022.9754037.

[10] Amit Kumar Gupta, Priya Mathur, Carlos M. Travieso-Gonzalez, Muskan Garg, and Dinesh Goyal. 2022. ORR: Optimized Round Robin CPU Scheduling Algorithm. In Proceedings of the International Conference on Data Science, Machine Learning and Artificial Intelligence (DSMLAI '21'). Association for Computing Machinery, New York, NY, USA, 296–304. https://doi.org/10.1145/3484824.3484917

[11] W. -C. Fan, C. -S. Wong, W. -K. Lee and S. -O. Hwang, "Comparison of Interactivity Performance of Linux CFS and Windows 10 CPU Schedulers," 2020 International Conference on Green and Human Information Technology (ICGHIT), Hanoi, Vietnam, 2020, pp. 31-34, doi: 10.1109/ICGHIT49656.2020.00014.

[12] I. Zagan and V. G. Gaitan, "Improving the performance of Real-Time Event Processing based on Preemptive Scheduler FPGA Implementation," 2020 International Conference on Development and Application Systems (DAS), Suceava, Romania, 2020, pp. 49-55, doi: 10.1109/DAS49615.2020.9108930.

[13] Giridas, K. & Nargunam, Shajin. (2012). Compatibility of Hybrid Process Scheduler in Green IT Cloud Computing Environment. International Journal of Computer Applications. 55. 27-33. 10.5120/8753-2649.

[14] Shakti Mishra, Dharmender Singh.Kushwaha et. Al., 2010. "An Efficient Job Scheduling Technique in Trusted Clusters for Load Balancing", The First International Conference on Cloud Computing, GRIDs, and Virtualization

[15] S. Z. Sheikh and M. A. Pasha, "Energy-Efficient Cache-Aware Scheduling on Heterogeneous Multicore Systems," in IEEE Transactions on Parallel and Distributed Systems, vol. 33, no. 1, pp. 206-217, 1 Jan. 2022, doi: 10.1109/TPDS.2021.3090587.

[16] Sergey Zhuravlev, Juan Carlos Saez, Sergey Blagodurov, Alexandra Fedorova, and Manuel Prieto. 2012. Survey of scheduling techniques for addressing shared resources in multicore processors. ACM Comput. Surv. 45, 1, Article 4 (November 2012), 28 pages. https://doi.org/10.1145/2379776.2379780

[17] Giang Son Tran, Thi Phuong Nghiem, Tuong Vinh Ho, and Chi Mai Luong. 2016. Extended process scheduler for improving user experience in multi-core mobile systems. In Proceedings of the 7th Symposium on Information and Communication Technology (SoICT '16). Association for Computing Machinery, New York, NY, USA, 417–424. https://doi.org/10.1145/3011077.3011106

[18] X. Gao and M. Qiu, "Efficient Process Scheduling for Multi-core Systems," 2022 IEEE 8th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS), Jinan, China, 2022, pp. 113-118, doi: 10.1109/BigDataSecurityHPSCIDS54978.2022.00030.

[19] T. Yu et al., "Collaborative Heterogeneity-Aware OS Scheduler for Asymmetric Multicore Processors," in IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 5, pp. 1224-1237, 1 May 2021, doi: 10.1109/TPDS.2020.3045279.

[20] Muneeswari, G. & Shunmuganathan, K.. (2011). A Novel Hard-Soft Processor Affinity Scheduling for Multicore Architecture using Multiagents. European Journal of Scientific Research. 55.

[21] S. Mohan, M. K. Yoon, R. Pellizzoni and R. Bobba, "Real-Time Systems Security through Scheduler Constraints," 2014 26th Euromicro

Conference on Real-Time Systems, Madrid, Spain, 2014, pp. 129-140, doi: 10.1109/ECRTS.2014.28.

[22] K. Guha, S. Saha and K. McDonald-Maier, "SENAS: Security driven ENergy Aware Scheduler for Real Time Approximate Computing Tasks on Multi-Processor Systems," 2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS), Torino, Italy, 2022, pp. 1-5, doi: 10.1109/IOLTS56730.2022.9897811.

[23] J. Dahlstrom and S. Taylor, "Migrating an OS Scheduler into Tightly Coupled FPGA Logic to Increase Attacker Workload," MILCOM 2013 - 2013 IEEE Military Communications Conference, San Diego, CA, USA, 2013, pp. 986-991, doi: 10.1109/MILCOM.2013.171.