A Comparative Analysis of HPC Scheduling Algorithms Based on Performance and Fairness

Sifat Tanvir
Dept of Computer Science and Engineering
Brac University
Dhaka, Bangladesh
sifattanvirsami@gmail.com

Md Tawhid Anwar
Dept of Computer Science and Engineering
Brac University
Dhaka, Bangladesh
write2tawhid@gmail.com

Zaber Mohammad
Dept of Computer Science and Engineering
Brac University
Dhaka, Bangladesh
zaber.mohammad@g.bracu.ac.bd

Sumaiya Tanjil Khan
Dept of Computer Science and Engineering
Brac University
Dhaka, Bangladesh
sumaiya.tanjil.khan@g.bracu.ac.bd

Rakin Bin Rabbani
Dept of Computer Science and Engineering
Brac University
Dhaka, Bangladesh
rakin.bin.rabbani@g.bracu.ac.bd

Md Sabbir Hossain
Dept of Computer Science and Engineering
Brac University
Dhaka, Bangladesh
md.sabbir.hossain1@g.bracu.ac.bd

Annajiat Alim Rasel
Dept of Computer Science and Engineering
Brac University
Dhaka, Bangladesh
annajiat@gmail.com

***Abstract*—This paper compares and analyzes the performance and fairness of several scheduling algorithms in HPC. The performance measurement matrices used are throughput, CPU utilization, scalability, turnaround time, waiting time, response time, overhead etc. The fairness comparison was made based on two conditions: Temporal starvation and resource underutilization. This paper also discusses the potential reasons behind certain algorithms performing worse or being more unfair than the others.**

***Index Terms*—HPC, FCFS, SJF, SRTF, RR, GE, ACO**

I. Introduction

High Performance Computing (HPC) is a fast growing field with huge potential in different sectors of computer science. Because of its advantages, contribution of the HPC environment is reaching not only in scientific developments but also in economic sectors. For developing a high performing environment, HPC systems combine integrated hardware, software and network [1], [2]. HPC market is going to have a CAGR of 9.44% over the period of 2021 - 2026 [3].

Due to its nature, HPC depends on scheduling tasks in a queuing mechanism where tasks are stored until they are executed [4]. The idea of HPC mainly depends on the interconnected HPC nodes where these nodes perform tasks and interact with each other for completing a total process. Studies on different scheduling algorithms have been done by different authors in HPC environments [5]. All these studies mainly focus on finding a best suited scheduling algorithm in order to develop an optimal performing environment.

Scheduling algorithms are very crucial for HPC algorithms because without proper scheduling algorithms implementation, the whole process may degrade its performance. To illustrate this, suppose in a system where complex and longer processing tasks can wait until finish of shorter tasks, Shortest Job First (SJF) algorithms will be a wise decision as this algorithm fulfills this criteria. If someone implements the traditional First Come First Serve (FCFS) algorithm there, it will degrade the overall performance drastically. On the other hand, if a system continuously requires processing of shorter tasks and hardly generates longer and complex tasks, implementation of SJF in such a scenario will not be a wise choice as it will cause the complex task to take a very long time. As a result, we can say that finding suited scheduling algorithms for HPC systems is a very delicate task.

In this paper, we mainly focus on different scheduling algorithms and their performance based on different matrices that have been described here. Section 2 explains the literature review and Section 3 focuses on the methodology. Performance and fairness analyzes of different scheduling algorithms are discussed in Section 4 and Section 5. Section 5 concludes the paper with conclusion and future work.

II. Related Works

In this section, we shall briefly go through the related conducted on different scheduling algorithms. Through the survey we have found some efficient algorithms that the authors have proposed based on the performance and fairness of HPC applications. The most important criteria to check are the hourly cost and performance since a fair comparison of providers and efficiency for cloud-based applications are made sure [6]. When selecting a provider to execute these applications, the behavioral nature of the target applications as well as the expected usage context should be taken into consideration. In paper [7] the authors discussed the critical problem of allocating the submitted tasks to heterogeneous servers in HPC systems. The authors have proposed a new deep reinforcement learning (DRL) enhanced greedy algorithm which consists of two-stage scheduling as in interacting task sequencing and task allocation. While comparing with the state-of-the-art baselines, their algorithm improves the system gain by about 10% to 30%.

In the cloud, HPC programs can face serious challenges that could undermine the gained benefits. To address these issues, [8] developed ASETS, or ”A SDN Empowered Task Scheduling System,” a novel approach for scheduling data-intensive High Performance Computing (HPC) jobs in a Cloud environment. While assigning tasks to virtual machines, SDN's ”bandwidth awareness” functionality is used to better utilize bandwidths. Energy conservation is becoming more critical in cloud computing, particularly when dealing with High Performance Computing (HPC). [9] developed a multi-objective genetic algorithm (MO-GA) for optimizing energy usage, CO2 emissions, and profit created by a geographically distributed cloud computing infrastructure. During the last few years, the scientific community has become increasingly interested in making efficient use of resources on a computing cloud. In terms of resource utilization and machine-level load balancing, they [10] introduced the RALBA (Resource-aware load balancing algorithm), which outperformed and appeared to be the optimal balance between complexity and performance. This research [11] proposes a genetic algorithm scheduling model based on a comparison of three existing methods for evaluating the quality of assigned tasks by users: round-robin, load index-based, and ABC-based job scheduling models.

[12] introduced the Coloured Petri Net (CPN) for hardware-software architecture, which contains the infrastructure for adaptive load balancing at two levels (node and inter-node). The model was used to design a dynamic distributed algorithm for the scheduling problem. Improvement of application latency in clouds has become an important concern among the researchers. In one paper [13], they proposed an adaptive and fast online application runtime prediction model known as State Space Models in order to improve the latency of HPC applications.

III. Algorithms

Which process will be executed first by the CPU while other processes will be on waiting state is decided by different CPU scheduling algorithms. Some of the scheduling is discussed below:

First Come First Serve (FCFS): FCFS, or First Come First Serve, is one of the simplest and most straightforward operating system scheduling algorithms. It works with the principle of executing chained requests and procedures in the sequence that they come or according to the request's arrival time. In every case, the algorithm processes the task which has been arrived at first. In this type of method, CPU gives the allocation to those processes which requests the CPU first. A FIFO (First In First Out) queue is used to manage this. When processes arrived they are stored at the end of the queue. As each process finishes its tasks, the next process gets picked from the start of the queue [14]. Both non-preemptive and pre-emptive scheduling algorithms are supported by FCFS.This algorithm is very advantageous as the procedure can be implemented very easily and it is very user friendly as well. [15]. On the other hand, since it is a Non-Preemptive CPU scheduling algorithm, so whenever the process gets the allocation from the CPU it will occupy the CPU until the executiion is finished. Moreover, the waiting time is also high on average. In terms of time sharing system, FCFS will not be an ideal choice and due to it's intelligibility, it is not efficient either.

Shortest Job First (SJF): This algorithm is a modified algorithm of FCFS. The main difference with the FCFS algorithm is that this algorithm executes the tasks which require the shortest time first while maintaining the queue. Shortest Job First (SJF) is a well-known CPU processing method. Preemptive and non-preemptive are the only two types of Shortest Job First Algorithms. When a new process enters, preemptive SJF allows the currently running process to be interrupted. When using a non-preemptive algorithm, the algorithm will continue to run even if a new process enters [16]. A unit of time is assigned to each job to complete it's task. Those batch processing which does not prioritize waiting time for jobs to finish is appropriate for this algorithm.For long terms scheduling SJF is mostly used and it's average waiting time is less in comparison to FIFO algorithm [14]. The SJF technique assigns the shortest average waiting time for a given set of procedures and is likely the most efficient in terms of average turnaround time. As the length of the upcoming CPU burst is not predictable SJF cannot be implemented for the short term CPU scheduling SJF. This technique could result in extremely long turnaround times or perhaps starvation. [17].

Shortest Remaining Time First (SRTF): Among the two types of SJF scheduling algorithm Shortest Remaining Time First or SRTF is the preemptive type. After a particular time, the execution of the process can be terminated in SRTF. From the available list, the process with the shortest remaining burst time is scheduled by the short term scheduler and executes the process as it arrives. SRTF is the best option since it ensures the shortest average waiting time. It serves as a benchmark for other algorithms because no other algorithm outperforms it [18]. It cannot be applied in practice because the burst time of the processes cannot be predicted ahead of time. The processes which have longer burst time starve due to this algorithm. Processes cannot have priorities defined for them. The reaction time of processes having a longer burst time is poor.

Round Robin (RR): This algorithm is similar to the FCFS algorithm. However, it switched among tasks which makes it different from the FCFS algorithm. It creates a time quantum and works for that time quantum on a specific task. After that it switches to another task. It is frequently used for multitasking as it is the simplest scheduling algorithm in nature [19]. Round robin is preemptive and works with the principle of time quantum or time slice as in the CPU allocates the resources to the next process after fixed interval time and a context change takes place. The process that gets preempted is appended to the end of the queue. It is a clock-driven hybrid model [20]. Some of the benefits of Round robin include the fact that it does not suffer from starvation or the convoy effect. In this algorithm all jobs receive a significant allocation of CPU and it handles all procedures equally. The processor output, on the other hand, will be lowered if the OS's slicing time is short. This approach takes longer to switch contexts, and its performance is highly dependent on time quantum. In addition, unlike other scheduling algorithms, we are unable to define priority.

Genetic Algorithm (GA): Genetic Algorithms (GA) is a kind of search algorithm with a high degree of randomness that is developed on the theory of bio genetics [21]. This algorithm uses random selection for optimal solution following heuristic search. Here priorities are defined by the genetic algorithm [22]. This algorithm performs very well in very large states and it is not only faster but also more efficient as compared to the conventional methods. However, this algorithm can not always ensure optimal answers.If it is not implemented properly, it may not provide the optimal or best solution.

Ant Colony Optimization (ACO): In fields such as computer operations research, the ant colony optimization algorithm (ACO) is a probabilistic method to addressing computing problems that can be simplified to finding optimal paths across networks. It operates on the basis of input and output. The ant colony algorithm may be performed indefinitely and adapt to changing conditions in real time. When the graph changes dynamically, they have an advantage over genetic algorithm approaches of similar problems. Due of its advantages, the ACO approach is frequently employed to solve combinatorial optimization problems [23].

IV. Performance Analysis

In the High Performance Computing (HPC) environment, scheduling algorithms are considered as one of the most important factors. While operating, HPC clusters distribute workloads among themselves maintaining dependency with each other. In order to ensure smooth performance, scheduling tasks among the HPC nodes using proper task scheduling algorithms is very important. Different scheduling algorithms have different pros and cons and based on the scenario different algorithms are necessary. However, a set of performance matrices can be declared to analyze and compare the performance of different scheduling algorithms [24].

A. Matrices

- Throughput: It means how many tasks are completed in a unit of the time. Complex tasks inherently require higher throughput. Maximum throughput means a better algorithm.
- CPU Utilization: Ensuring 100% utilization while completing a CPU cycle. Maximum utilization is better.
- Turnaround Time: Time required for a process's initialization to submission. Minimum turnaround time is better for an algorithm.
- Scalability: It refers to the system's capacity to complete its high performing activities. High scalability refers to the better scheduling algorithm.
- Waiting Time: It means the time spent in the waiting queue by a process. Minimum waiting time means better performance.
- Response Time: It refers to the time between first response and task submission time. Minimum response time is desirable for a system.
- Overhead: The time spent while the CPU switches from one task to another. Low overhead is better for performance.

B. Analysis

In terms of the matrices, the FCFS algorithm has maximum response time, waiting time, and overhead. On the other hand, it has minimum throughput, CPU utilization and turnaround time. The RR algorithm has maximum overhead, response time, waiting time, fairness and throughput. It has minimum response time and CPU Utilization. GA has maximum throughput and minimum overhead and turnaround time, however cannot assure the production of exact optimum output [26]. SJF algorithm has maximum overhead, turnaround time and throughput and minimum waiting time.

From the above analysis, it can be seen that different algorithms have different approaches and according to their approaches they have different characteristics in terms of matrices. However, we can not decide which algorithm is best in all scenarios and the reason is that not a single algorithm can be declared as the best algorithm. Based on the scenario certain algorithms perform better than other algorithms. To illustrate, in a HPC environment where different interconnected nodes share information with each other Round Robin can be a better option than First Come First Serve. The reason behind this is that the Round Robin algorithm switches between tasks after a certain time quantum and so interdependent nodes can receive a prerequisite tasks result earlier than the FCFS approach. Although each algorithm has its own merits and demerits, based on the performance matrices, some algorithms such as RR and GE generally perform better than other algorithms.

V. Fairness Analysis

How workloads are divided among the CPUs is known as fairness. Equal fairness is desired in an algorithm. Computing environments that are shared among many users are expected to distribute the available resources in such a manner that no user gets deprived of the resources they were supposed to have, resulting in overall maximum satisfaction. However, in reality, one or more users can act selfishly to monopolize the resources which leads to undesirable delays in completing tasks of the users [25]. When processes are stuck in queue, it causes temporal starvation. Again, if processes unnecessarily holding resources with high computational power leads to underutilization of resources. Both scenarios minimize satisfaction and induce unfairness. The fairness analysis of several algorithms have been summarized in table I.

TABLE I
FAIRNESS ANALYSIS

Algorithm	Probability of Temporal Starvation	Probability of Resource Underutilization	Reason of Unfairness
FCFS	High	High	Shorter process forced to be stuck behind large processes, also known as Convoy Effect
SJF	High	High	High priority jobs may be stuck in queue due to higher burst time
SRTF	High	High	High priority jobs may be stuck in queue due to higher remaining time
RR	Depends on Time Quantum	Depends on Time Quantum	Too low time quantum can cause resource underutilization, too high time quantum can cause higher priority tasks to starve
GA	Lower	Lower	Improper designing of the objective function, representation and operators
ACO	Minimum	Minimum	Excessive convergence time may increase overall waiting time for all processes

None of the algorithms are immune to temporal starvation or resource underutilization. However, the probability varies among the algorithms. FCFS has a high affinity of encountering starvation, increased average waiting time and underutilization because of the convoy effect [26]. SJF and SRTF prevent the convoy effect, but do not take priority of a task into account. Therefore, shorter tasks get executed despite having low priority and increases starvation excessively [26]. A perfect time quantum in RR may come very close to achieving maximum satisfaction, but it is highly experimental and unlikely. GA is good at finding optimized solutions compared to the aforementioned algorithms. The progress of GA is based on the fitness value calculated on each population generation. The process continues until a satisfactory solution is achieved which makes this algorithm computationally expensive. However a trade-off between computational expense and finding the most suitable resource distribution scheme can prove to be highly effective in achieving maximum satisfaction. ACO is also very promising when it comes to resource utilization [26]. If the convergence time is not excessive, it is highly likely to achieve maximum satisfaction and minimum starvation.

VI. Conclusion and future Work

Based on the overall analysis it is evident that how an algorithm will behave completely depends on the scenarios and computational power. We can assume that RR, GE or ACO algorithms have the potential to perform better than other algorithms and achieve maximum satisfaction if the time quantum is guessed properly and objective function is designed properly. Therefore, if enough computational power can be ensured and possibility of failure can be ignored, GE or ACO is preferred over RR. Otherwise, RR is a safer option. However, some algorithms can be modified to achieve higher satisfaction as shown in [27]–[29]. We aim to include modified scheduling algorithms in our future analysis as well. We also want to dive deep into finding ways of guessing perfect time quantum and designing proper objective function.

References

[1] F. Pinel, J. E. Pecero, S. U. Khan, and P. Bouvry, Energy-efficient scheduling on milliclusters with performance constraints, in Proceedings of the 2011 IEEE/ACM International Conference on Green Computing and Communications, IEEE Computer Society, Washington, DC, 2011, pp. 44–49.

[2] L. Wang, S. U. Khan, D. Chen, J. Koodziej, R. Ranjan, C.-z. Xu, and A. Zomaya, Energy-aware parallel task scheduling in a cluster, Future Generation Computer Systems, 29 (2013), pp. 1661–1670.

[3] Mordorintelligence.com. 2022. High-Performance Computing Market — 2021 - 26 — Industry Share, Size, Growth - Mordor Intelligence. [online] Available at: ¡https://www.mordorintelligence.com/industry-reports/high-performance-computing-market¿ [Accessed 13 January 2022].

[4] D. Klusacek ´ , Scheduling in grid environment, Ph.D. thesis, Masaryk University, Brno, Czech Republic, 2008.

[5] Balis, B., Figiela, K., Jopek, K., Malawski, M., Pawlik, M. (2017). Porting HPC applications to the cloud: A multi-frontal solver case study. Journal of Computational Science, 18, 106– 116.

[6] Roloff, E., Diener, M., Carissimi, A. (2012). High performance computing in the cloud:Deployment, performance and cost efficiency. In 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom). Piscataway, NJ: IEEE.

[7] Y. Yang and H. Shen, ”Deep Reinforcement Learning Enhanced Greedy Algorithm for Online Scheduling of Batched Tasks in Cloud in Cloud HPC Systems,” in IEEE Transactions on Parallel and Distributed Systems, doi: 10.1109/TPDS.2021.3138459.

[8] S. Jamalian and H. Rajaei, ”Data-Intensive HPC Tasks Scheduling with SDN to Enable HPC-as-a-Service,” 2015 IEEE 8th International Conference on Cloud Computing, 2015, pp. 596-603, doi: 10.1109/CLOUD.2015.85.

[9] Y. Kessaci, N. Melab and E. Talbi, ”A pareto-based GA for scheduling HPC applications on distributed cloud infrastructures,” 2011 International Conference on High Performance Computing Simulation, 2011, pp. 456-462, doi: 10.1109/HPCSim.2011.5999860.

[10] A. Hussain, M. Aleem, M. A. Islam and M. A. Iqbal, ”A Rigorous Evaluation of State-of-the-Art Scheduling Algorithms for Cloud Computing,” in IEEE Access, vol. 6, pp. 75033-75047, 2018, doi: 10.1109/ACCESS.2018.2884480.

[11] Jang, S. H., Kim, T. Y., Kim, J. K. (2012). The study of genetic algorithm-based task scheduling for cloud computing. International Journal of Control and Automation, 5(4), 157–162.

[12] I. Dan Mironescu and L. Vinţan, ”A task scheduling algorithm for HPC applications using colored stochastic Petri Net models,” 2017 13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP), 2017, pp. 479-486, doi: 10.1109/ICCP.2017.8117051.

[13] M. Naghshnejad and M. Singhal, ”Adaptive Online Runtime Prediction to Improve HPC Applications Latency in Cloud,” 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), 2018, pp. 762-769, doi: 10.1109/CLOUD.2018.00104.

[14] Bibu, G. D., Nwankwo, G. C. (2019). Comparative analysis between first-come-first-serve (FCFS) and shortest-job-first (SJF) scheduling algorithms. Int J Comput Sci Mobile Comput, 8(5), 176-181.

[15] Schwiegelshohn, U., Yahyapour, R. (1998, January). Analysis of first-come-first-serve parallel job scheduling. In SODA (Vol. 98, pp. 629-638).

[16] Putra, T. D. (2020). Analysis of Preemptive Shortest Job First (SJF) Algorithm in CPU Scheduling. International Journal of Advanced Research in Computer and Communication Engineering, 9(4).

[17] Hamayun, M., Khurshid, H. (2015). An optimized shortest job first scheduling algorithm for CPU scheduling. J. Appl. Environ. Biol. Sci, 5(12), 42-46.

[18] Aboalama, M., Yousif, A. (2015). Enhanced job scheduling algorithm for cloud computing using shortest remaining job first. International Journal of Computer Science Management Studies, 15(6), 65-68.

[19] Fürnkranz, J. (2002). Round robin classification. The Journal of Machine Learning Research, 2, 721-747.

[20] Yadav, R. K., Mishra, A. K., Prakash, N., Sharma, H. (2010). An improved round robin scheduling algorithm for CPU scheduling. International Journal on Computer Science and Engineering, 2(04), 1064-1066.

[21] Xu, J. (2021). Improved Genetic Algorithm to Solve the Scheduling Problem of College English Courses. Complexity, 2021.

[22] Gonçalves, J. F., de Magalhães Mendes, J. J., Resende, M. G. (2005). A hybrid genetic algorithm for the job shop scheduling problem. European journal of operational research, 167(1), 77-95.

[23] Deng, W., Xu, J., Zhao, H. (2019). An improved ant colony optimization algorithm based on hybrid strategies for scheduling problem. IEEE access, 7, 20281-20292.

[24] Jyotirmay Patel, A.K.Solanki, 2012, Performance Evaluation of CPU Scheduling by Using Hybrid Approach, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH TECHNOLOGY (IJERT) Volume 01, Issue 04 (June 2012),

[25] Sedighi, Art Deng, Yuefan Zhang, Peng. (2014). Fairness of Task Scheduling in High Performance Computing Environments. Scalable Computing: Practice and Experience. 15. 273-285. 10.12694/scpe.v15i3.1020.

[26] Razzaq S. et al. (2019) Scheduling Algorithms for High-Performance Computing: An Application Perspective of Fog Computing. In: Jan M., Khan F., Alam M. (eds) Recent Trends and Advances in Wireless and IoT-enabled Networks. EAI/Springer Innovations in Communication and Computing. Springer, Cham.

[27] Aravind, Alex. (2013). Simple, space-efficient, and fairness improved FCFS mutual exclusion algorithms. Journal of Parallel and Distributed Computing. 73. 1029–1038. 10.1016/j.jpdc.2013.03.009.

[28] H. B. Parekh and S. Chaudhari, ”Improved Round Robin CPU scheduling algorithm: Round Robin, Shortest Job First and priority algorithm coupled to increase throughput and decrease waiting time and turnaround time,” 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC), 2016, pp. 184-187, doi: 10.1109/ICGTSPICC.2016.7955294.

[29] Arya, Govind Nilay, Kumar Prasad, Dr. (2018). An Improved Round Robin CPU Scheduling Algorithm based on Priority of Process. International Journal of Engineering and Technology(UAE). 7. 238-241. 10.14419/ijet.v7i4.5.20077.