

Comparative Performance Analysis of Job Scheduling Algorithms in a Real-World Scientific Application

Fernando Emilio Puntel[1(✉)], Andrea Schwertner Charão[1(✉)], and Adriano Petry[2(✉)]

[1] Federal University of Santa Maria, Santa Maria, RS, Brazil
{fepuntel,andrea}@inf.ufsm.br
[2] National Institute for Space Research, Santa Maria, RS, Brazil
adriano.petry@inpe.br

Abstract. In High Performance Computing, it is common to deal with substantial computing resources, and the use of a Resource Management System (RMS) becomes fundamental. The job scheduling algorithm is a key part of a RMS, and the selection of the best job scheduling that meets the user needs is of most relevance. In this work, we use a real-world scientific application to evaluate the performance of 4 different job scheduling algorithms: First in, first out (FIFO), Shortest Job First (SJF), EASY-backfilling and Fattened-backfilling. These algorithms worked with RMS SLURM workload manager, considering a scientific application that predicts the earth's ionosphere dynamics. In the results we highlight each algorithm's strength and weakness for different scenarios that change the possibility of advancing smaller jobs. To deepen our analysis, we also compared the job scheduling algorithms using 4 jobs of Numerical Aerodynamic Sampling (NAS) Parallel Benchmarks in a controlled scenario.

Keywords: Job scheduling · High performance computing · SLURM and resource management system

1 Introduction

High performance computing environments are important in a variety of fields that require significant computational resources, like astronomy and meteorology. Among the possibilities of high performance environments, computer clusters are predominant due to their performance and scalability. Distributed computing resources available in a cluster require the use of Resource Management System (RMS) to optimize the distribution of jobs of applications. The main goal in a RMS applied to clusters is to coordinate the available computing resources so they work as a single computer, targeting the reduction of total processing time and idle resources. SLURM (Simple Linux Utility for Resource Management) is one of the most important RMS, and it is present in 6 from 10 top supercomputer list (TOP500). It is open source, fault tolerant and highly scalable [21,22].

© Springer Nature Switzerland AG 2020
O. Gervasi et al. (Eds.): ICCSA 2020, LNCS 12249, pp. 447–462, 2020.
https://doi.org/10.1007/978-3-030-58799-4_33

One of the main parts of RMS is the scheduling algorithm, responsible for defining when and at which processing node the jobs execute. The choice of a given scheduling algorithm must consider the application constraints and the available computing environment. Although several papers present original job scheduling procedures, that are evaluated and compared to others, evaluations are usually conducted on simulators, with synthetic workloads. This artificial basis can lead to inaccurate results for real applications. In this work, we report an evaluation of job scheduling algorithms in a real case application, using a scientific cluster running daily.

2 Job Scheduling

Job scheduling is an important research topic in HPC. The choice of the job scheduling approach can impact directly the application performance [7,16].

Usually, job scheduling aims to increase computational resource usage, which means the resources are used most of the period, and also submitted jobs finish within expected time [17]. Besides that, job scheduling can also target other goals, such as approximate the waiting time among jobs [19], reduce latency or response time, or limit the use of power resources [8,9]. Different goals may conflict, so the use of resources depends on the computing environment and application.

Several job scheduling algorithms are available. Among them, we have selected four job scheduling algorithms to perform the experiments. The Figs. 1, 2, 3 and 4 show a comparison between them in a hypothetical case of 4 jobs that require different resources and processing time, using a cluster composed of 3 processing cores, each one available at different moments.

The FIFO algorithm is a classic model that runs the jobs by queue order, i.e., the first job of the queue is the first submitted. Its main disadvantage is that it is not flexible to move forward smaller jobs to execute before. Figure 1 illustrates FIFO job scheduling for the hypothetical case.

The SJF (Shortest Job First) [12] algorithm is similar to backfilling algorithms [13]. It advances smaller jobs when there are available computational resources to run them, without other verification. Figure 2 shows the SJF addressing the hypothetical case.

The EASY-backfilling aims to advance smaller jobs provided it does not delay the first job of the queue [13]. The EASY-backfilling algorithm is the most popular job scheduling algorithm based in the conservative job scheduling backfilling optimization. Figure 3 shows the EASY-backfilling procedure for the hypothetical case.

The Fattened-backfilling algorithm is very similar to EASY-backfilling. The advance of smaller jobs is similar to EASY-backfilling, but in the calculation to choose the job to be advanced, it is considered the Average Waiting Time (AWT) coefficient of already finished jobs, what allows the advance of smaller jobs of queue [11]. Figure 4 illustrates the Fattened-backfilling execution.

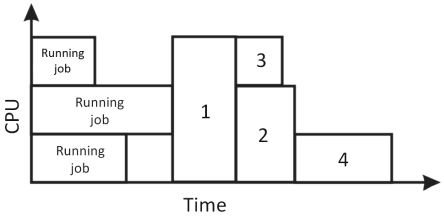

Fig. 1. FIFO Job scheduling algorithm in hypothetical cases

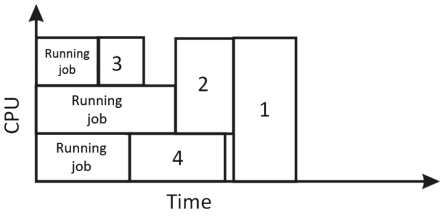

Fig. 2. SJF Job scheduling algorithm in hypothetical cases

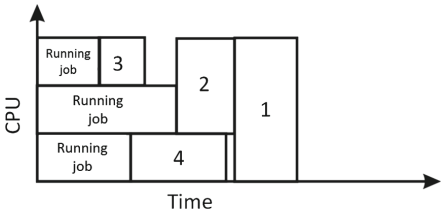

Fig. 3. EASY-backfilling Job scheduling algorithm in hypothetical cases

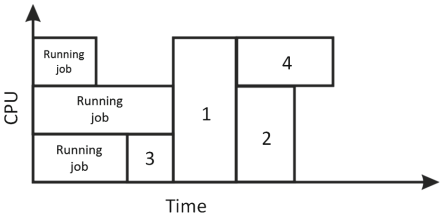

Fig. 4. Fattened-backfilling Job scheduling algorithm in hypothetical cases

3 Ionosphere Forecasting

The real application used in experiments is based on forecasting the Earth ionosphere dynamics using a mathematical model [14]. The system runs daily and provides total electron content (TEC) maps for every hour in South America region. These maps are available at http://www2.inpe.br/climaespacial/portal/tec-supim-prevision/. The application first step is to run the physical model, that is an enhanced version of the Sheffield University Plasmasphere-ionosphere Model (SUPIM), adapted to South America singularities. The model considers several chemical and physical interactions among particles in ionosphere, and provides discreet outputs (ions and electronic concentrations) along geomagnetic field lines. These outputs are then processed to the Data Assimilation and Visualization System (DAVS). DAVS is able to modify the model outputs considering observational data, in order to adjust the simulation to the available measurements. Besides that, it provides estimatives of electron concentrations in homogeneous grid, that are summed in height to result in TEC values in a 2-dimensional map.

4 Experimental Evaluations

The experiments have been performed in a CRS-INPE (Southern Regional Space Research Center, National Institute for Space Research) computing cluster. The cluster is composed of 5 processing nodes, each one consisting of with 8 Intel Xeon

E5-2609 2.40 GHz processors containing 4 cores, and 72 GB of RAM memory. We used SLURM and two databases in MySQL to manage the resources and jobs.

SLURM was installed and configured in all processing nodes and the controlling node of a cluster. The configuration file holds the cluster architecture information. By default, SLURM is configured so every processing node is unavailable for other tasks during the processing of a job or a group of jobs. This configuration was changed as follows, to make it possible the processing nodes to run different jobs simultaneously, where the jobs are allowed to ask for a number of CPUs, processing cores or amount of memory to run.

- *SelectType*: this option is by defalut set as *select/linear*, where only one job can run in a node. For the experiments, this option was defined as *SelectType=select/cons_res*, so the node can be shared with other jobs;
- *SelectTypeParameters*: this option is added to SLURM configuration file when *SelectType* is changed. For the experiments, this option was defined as *CR_CPU_Memory*, where the jobs can request CPU and memory;
- *DefMemPerCPU*: this option is added to define a memory to reserve if the job does not specify a memory request. In the experiments, it was set to 5000 MB;

We first compared the performance of the job scheduling algorithms under three different scenarios using SUPIM-DAVS jobs: intermediate, favorable and unfavorable cases. In each case, the jobs are organized in queue for job scheduling algorithms evaluation, and executed 30 times.

For the evaluations, we selected 56 SUPIM jobs, where each one represent a different longitude in the South America area (ranging from -35°W to -85°W), and 24 DAVS jobs, corresponding to every hour in a day. The selected day simulated by SUPIM-DAVS jobs was January, 29th 2015, when the solar flux (F10.7) was 123.73 sfu (solar flux units) [4].

The resource requests for the jobs are shown in Table 1. These requests were defined before jobs execution. The CPU requisition of 0.5 in DAVS jobs is because it is possible to run two DAVS jobs per CPU.

Table 1. The requisitions SUPIM and DAVS jobs

Job	CPU	Memory (MB)	Time of execution (sec)
SUPIM	1	3072	3060
DAVS	0.5	10240	1050

Since SUPIM forecasting jobs need a pre-execution task, composed of file downloads and initial setup, the monitors of CPU usage, memory usage and average waiting time were initialized at the moment that the first job was submitted. Considering the cluster is dedicated only for the execution these experiments, it is guaranteed that the measurements correspond to the jobs under evaluation.

Besides SUPIM job execution experiments, we also evaluated the job scheduling using four different types of jobs of NAS Parallel Benchmarks [2] in a controlled scenario. The aim was to investigate the difference in results from real to controlled scenarios, where a significant mismatch could indicate alternative problems.

Since job scheduling algorithms used in experiments rely on information about the required processing time for each task, we had to estimate it. This estimative could be assigned to the user during submission process. However, considering that in most cases users overestimate such time [1,15], afraid of job interruption during processing – leaving computational resources idle [10,20], this approach is unlikely to succeed. A better choice would be to update the processing mean time for a given task considering its last few executions [18]. In this work the mean time is calculated over the last five executions of each task.

The CPU and memory use was checked using the tools *mpstat* [5,6] and *vmstat* [6], respectively. For execution time and mean waiting time measurements, Linux library *time* was used.

4.1 Intermediate Scenario

Three of the job scheduling algorithms in experiments may advance jobs execution in queue. Depending on the algorithm outcome, the queue can be organized differently. For this scenario, in some cases a job scheduling algorithm can advance smaller jobs. In other cases the scheduling algorithm behave like FIFO. Figure 5 illustrates a Gantt graph in a possible intermediate scenario, where we can observe some jobs are executing, while other jobs are waiting for resources in queue.

In scenario present in Fig. 5 the job scheduling algorithm will decide:

- after completion of the two DAVS jobs (currently running), DAVS jobs in queue (first and second positions) will be submitted, and when they finish the third job in queue (DAVS) is submitted;
- the first job in queue will be SUPIM (currently fourth in queue), and it is expected to start running after completion of a SUPIM job currently in execution. At this point, the job scheduling algorithm may decide to advance or not the seventh job in queue (DAVS) to start running with the third in queue (DAVS), because this advance will not delay SUPIM job;
- not to advance the other jobs in queue.

Figures 6 and 7 show respectively the application makespan and CPU use for SUPIM and DAVS jobs using different job scheduling. In this case, it is possible to observe that FIFO job scheduling had a worse performance compared to the others, while the other job scheduling presented similar results.

Figure 8 shows the average waiting time for jobs in the queue. It can be observed how FIFO algorithm had significant worst result when compared to other algorithms. Considering EASY-backfilling and Fattened-backfilling use waiting time statistics to estimate the coefficient of smaller jobs advance, both had a similar performance.

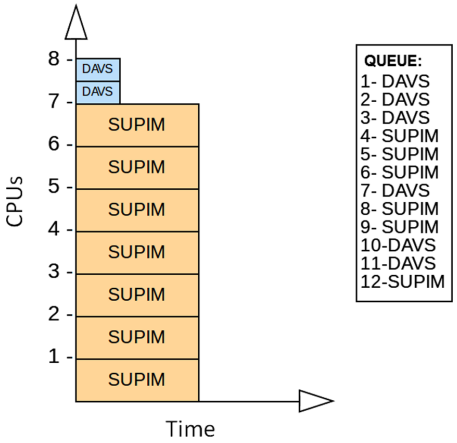

Fig. 5. Intermediate scenario

Table 2. Intermediate scenario waiting time for SUPIM and DAVS jobs

Job Scheduling Algorithm	All Jobs (sec)	SUPIM (sec)	DAVS (sec)
FIFO	1244.31	1328.75	1205.82
SJF	844.26	1178.16	202.46
EASY-backfilling	1045.36	1145.1	940.63
Fattened-backfilling	1049.06	1131.33	932.3

Table 2 depicts the average waiting time of Fig. 8 for SUPIM and DAVS jobs separately. For DAVS jobs, SJF shows a significant reduction in waiting time, since it can advance more DAVS jobs in comparison to other algorithms.

In general, for the intermediate scenario, FIFO algorithm had an overall inferior performance, while other job scheduling algorithms presented a similar

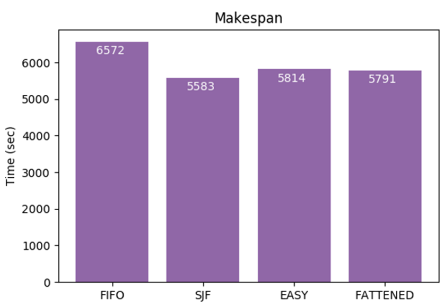

Fig. 6. Application makespan in intermediate scenario

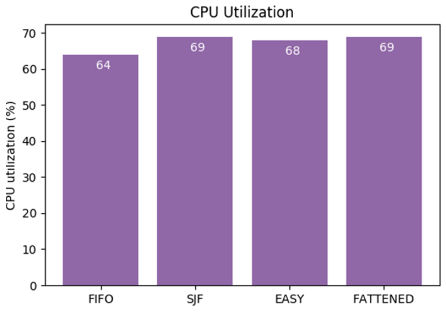

Fig. 7. CPU use in intermediate scenario

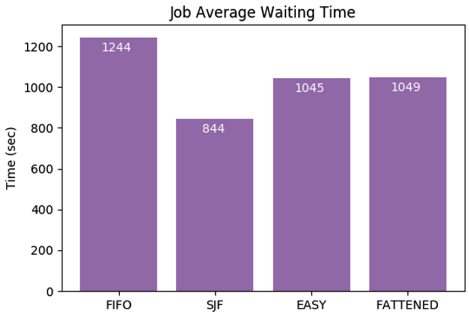

Fig. 8. Average waiting time in intermediate scenario

performance in all metrics. Despite EAS-backfilling and Fattened-backfilling use different approach than SJF to advance small jobs, these algorithms had a similar performance, including makespan and CPU use measurements.

4.2 Favorable Scenario

In the favorable scenario, the jobs are organized in queue to increase the advance of smaller jobs. Here, SJF, EASY-Backfilling and Fattened-backfilling performance is expected to improve. Figure 9 shows a possible favorable scenario Gantt graph, with some jobs executing and other waiting in queue.

After completion of running jobs, scheduling algorithms will decide:

– after completion of the two DAVS jobs (currently running), DAVS jobs in queue (first and second positions) will be submitted, and when they finish the third job in queue (DAVS) is submitted;

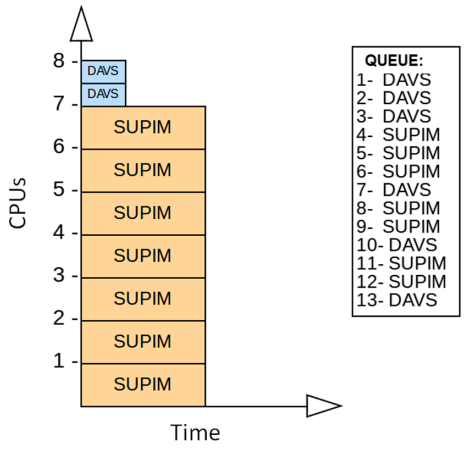

Fig. 9. Favorable scenario

– the first job in queue becomes SUPIM (currently fourth in queue), but in the lack of resources, the job scheduling algorithms that can advance small jobs will advance the seventh job in queue (DAVS) to start running with the third in queue (DAVS), because this advance will not delay SUPIM job;
– after completion of the seven SUPIM jobs (running initially), the SUPIM jobs in queue (positions 4th, 5th, 6th, 8th, 9th and 11th) are submitted, as well as DAVS job (position 10th in queue);
– since the next job in queue is SUPIM (position 12th), waiting to start after the DAVS job running in CPU 7 or 8, it is possible to advance DAVS job (position 13th in queue), because this advance will not delay SUPIM job.

Figure 10 shows the application makespan for the job scheduling algorithms. Like in the intermediate scenario, the FIFO algorithm had the worst results when compared to the other algorithms.

The Fig. 11 present CPU use. It is possible to observe that the FIFO presented inferior performance while the other algorithms had similar performances.

Then we compare EASY-backfilling and Fattened-backfilling in detail (see Table 3), it is possible to observe that Fattened-backfilling is able to advance more jobs and achieved a better result in DAVS jobs, and consequently increasing SUPIM jobs waiting time.

Table 3. Favorable scenario waiting time for SUPIM and DAVS jobs

Job scheduling algorithm	All Jobs (sec)	SUPIM (sec)	DAVS (sec)
FIFO	1218.43	1436.7	878.26
SJF	825.90	1171.16	146.26
EASY-backfilling	883.7	1100.53	550.65
Fattened-backfilling	890.90	1125.74	489.67

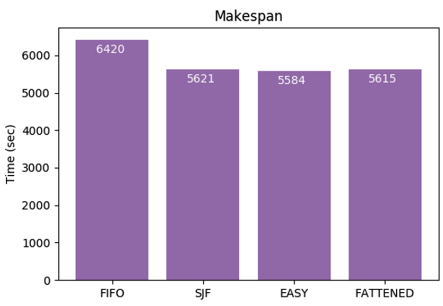

Fig. 10. Application makespan in favorable scenario

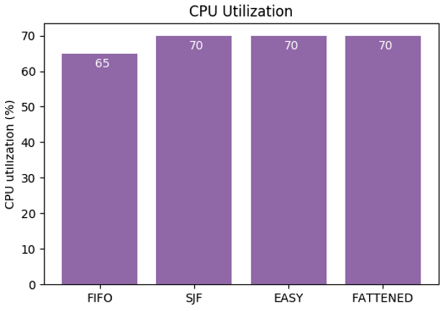

Fig. 11. CPU use in favorable scenario

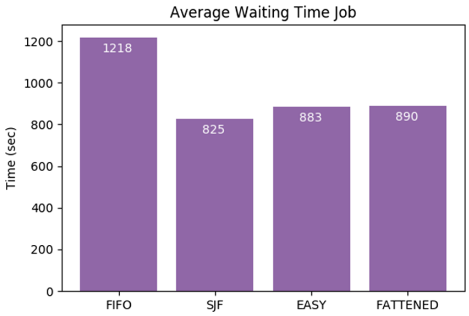

Fig. 12. Average waiting time in favorable scenario

4.3 Unfavorable Scenario

The queue in the unfavorable scenario is organized to difficult job scheduling to advance small jobs. In this scenario, regularly the processing nodes show intervals of idle resources. Figure 13 shows a Gantt graph for an hypothetical unfavorable scenario, with some jobs executing and other in queue.

After completion of running jobs, scheduling algorithms will decide:

- after completion of a DAVS job (currently running), the first in queue will be submitted (SUPIM job). The job scheduling algorithms that can advance DAVS jobs. (2nd in queue), since it would delay the first job in queue (SUPIM);
- after completion of SUPIM jobs, the jobs in queue will be submitted almost in order they are in queue, making it difficult to advance small jobs;
- such difficulty in advancing jobs is present through all the execution of the task.

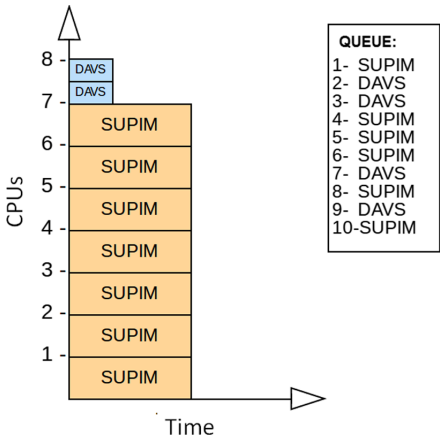

Fig. 13. Unfavorable scenario

In the unfavorable scenario, the FIFO algorithm showed again an inferior performance in makespan (see Fig. 14). In this scenario, the SJF performed better compared to EASY-Backfilling and Fattened-Backfilling. This is because even in an unfavorable scenario the SJF can perform the advance of smaller jobs as soon as some node have computational resources available.

Like in the other scenario, the CPU use presented similar result for SJF, EASY-backfilling and Fattened-backfilling algorithms, as show in Fig. 15. The job scheduling with inferior results was FIFO.

The average waiting time for the jobs in queue, shown in Fig. 16 and detailed in Table 4, presented the worse performance for FIFO. SJF was better than EASY-backfilling and Fattened-backfilling, which performed similarly despite Fattened-backfilling uses techniques to overcome the performance of EASY-backfilling, what did not hold in unfavorable environment.

Table 4. Unfavorable scenario waiting time for SUPIM and DAVS jobs

Job scheduling algorithm	All jobs (sec)	SUPIM (sec)	DAVS (sec)
FIFO	1218.43	1436.7	878.26
SJF	825.90	1171.16	146.26
EASY-backfilling	883.15	1100.53	550.65
Fattened-backfilling	890.90	1125.74	489.67

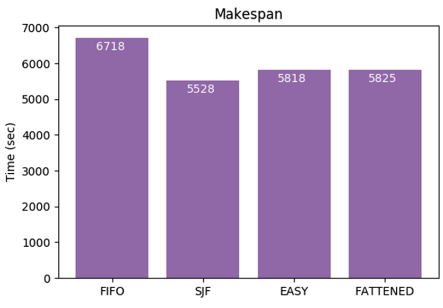

Fig. 14. Application makespan in unfavorable scenario

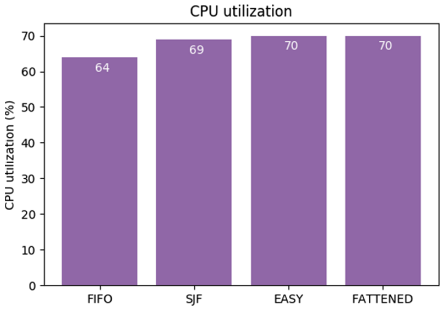

Fig. 15. CPU use in unfavorable scenario

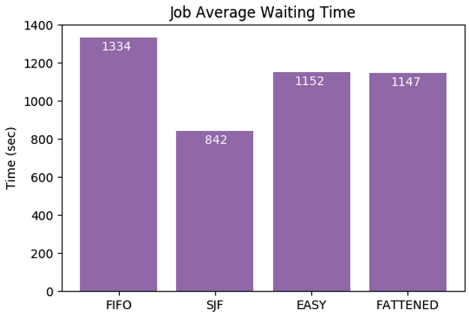

Fig. 16. Average waiting time in unfavorable scenario

4.4 Discussion

In all scenarios evaluated, the algorithm that presented worse performance was FIFO, since it does not allow the advance of jobs, leaving gaps of idle resources unused. This is why, in general, applications centred in the reduction of processing time hardly would benefit from choosing FIFO.

Other important issue to assess is the jobs' mean waiting time in queue. SJF benefits smaller jobs that are advanced, reducing its mean waiting time usually more than other algorithms. In favorable case, EASY-backfilling and Fattened-backfilling were able to achieve performances comparable to SJF, supporting they can effectively advance jobs. But in intermediate and unfavorable cases, SJF performed slightly better.

Although different scenarios were designed by changing the queue inputs in a scientific application, the results obtained were not significantly different from each other. Even FIFO, that showed the worse performance, did not compromise the ionospheric prediction results, since there were only 2 types of jobs, and the number of jobs to schedule was relatively low. So, it is also interesting to submit the scheduling algorithms to different and heterogeneous environments and observe the results.

4.5 Controlled Scenario

In this section we evaluate the algorithms in a controlled environment using different workloads. For experiments, NAS (Numerical Aerodynamic Sampling) Parallel Benchmarks (NPB) [2] jobs were chosen. NPB is a set of programs designed to evaluate the performance of parallel supercomputers. Basically, the benchmarks are derived from fluid dynamics applications [3]. The NPB has different applications, and for the experiments 4 different types were used:

- BT (Block Tri-diagonal solver), that performs the computational fluids processing;
- FT (Discrete 3D fast Fourier Transform): benchmark that solves 3D equations using spectral FFT;

– MG (Multi-Grid on a sequence of meshes): computes the solution of the Poisson equation;
– SP (Scalar Penta-diagonal solver): solves multiple independent systems of dominating equations diagonally and not diagonally.

The choice of the benchmarks and the requisitions of each job was done randomly to force a heterogeneous job scheduling. Table 5 presents the computational resources requested by each job after five executions.

Table 5. NPB Job Requirements

Job	CPU	Memory (MB)	Runtime (sec)
BT	2	1280	180
FT	2	5132	240
MG	4	26624	420
SP	6	5132	680

We run 110 jobs for the experiments: 34 BT, 34 FT, 25 MG and 17 SP jobs. The number of jobs with small requisitions was large, so the scheduling algorithms were able to perform the advance of a large number of jobs. Like in previous experiments, the cluster was used in a dedicated way and each algorithm was executed 30 times. For the BT, FT and MG jobs the standard input data was used. For the SP jobs, the number of iterations was changed to 160 and the problem size to $256 \times 256 \times 256$.

4.6 Results

Figure 17 illustrates the makespan when using the four algorithms in this case. It is possible to observe that the EASY-backfilling algorithm presented improvement in execution time by 10% when compared to the SJF algorithm. It seems that EASY-backfilling algorithm respected the execution queue and executed the largest jobs in its intended time, unlike the SJF algorithm, which advanced smaller jobs and left larger jobs to the end. As a result, larger jobs end up delaying execution. Also, in this scenario, the FIFO achieved better performance when compared to the SJF, precisely by running the larger jobs in their time and not delaying them at the beginning.

CPU and memory use are shown in Fig. 18 and 19, respectively. EASY-backfilling and Fattened-backfilling algorithms presented better performance for CPU use when compared to the other algorithms. EASY-backfilling was also superior to the other algorithms for memory use, since it tries to increase the use of resources during job scheduling. When compared to SJF, the result is about 20% better.

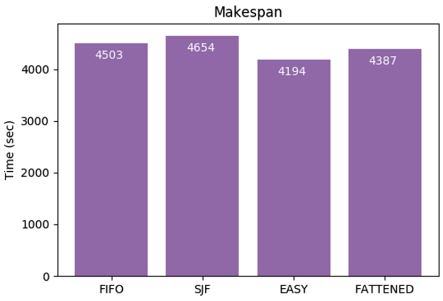

Fig. 17. Makespan in controlled scenario

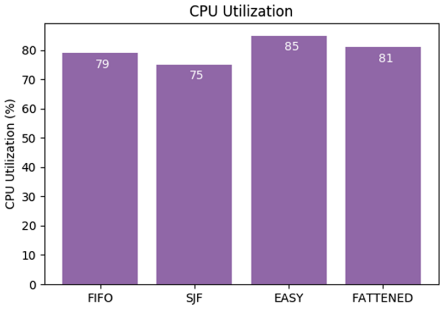

Fig. 18. CPU use in controlled scenario

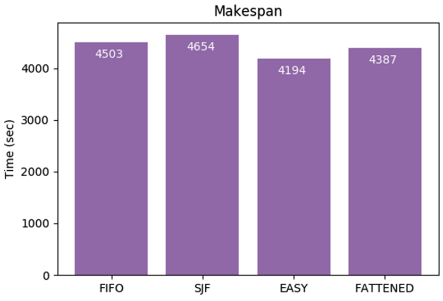

Fig. 19. Memory use in controlled scenario

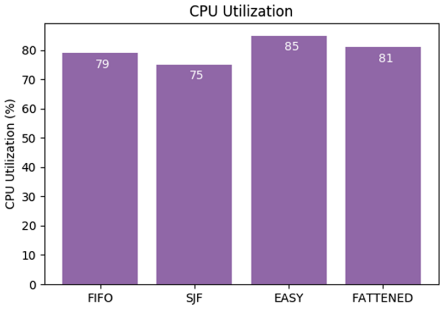

Fig. 20. Average waiting time in controlled scenario

Figure 20 shows the average waiting time in the job queue. In this case it is possible to observe that even if the SJF algorithm advances all the smaller jobs to execute first, the Fattened-backfilling algorithm obtains a superior performance, because it allows smaller jobs or smaller priorities to be advanced.

Table 6 shows the average total and individual waiting time. It is important to note that the SJF algorithm performed considerably better than EASY-backfilling only when compared to smaller jobs (FT and BT). The comparison of EASY-backfilling and Fattened-backfilling algorithms shows Fattened-backfilling obtained inferior performance only in the waiting time of SP jobs, precisely because the algorithm makes smaller jobs easier to advance.

In general, the FIFO algorithm presented inferior performance when compared to the other algorithms in this scenario. A point to be highlighted is the fact that the Fattened-backfilling algorithm obtained results similar to EASY-backfilling, although it does not target the improvement in computational performance metrics. The SJF algorithm was impaired in this experiment, because the delay of starting larger jobs caused a delay in finishing the task and consequently a decrease in the CPU use rate.

Table 6. Controlled scenario waiting time for jobs

Job scheduling algorithm	All jobs (sec)	SP (sec)	MG (sec)	FT (sec)	BT (sec)
FIFO	2712	1486	2134	3181	3360
SJF	1546	1966	1974	1253	1357
EASY-backfilling	1903	1432	2005	1907	2106
Fattened-backfilling	1399	1912	1657	1144	1246

5 Conclusions

This work evaluated the performance of different scheduling algorithms used with SLURM in a scientific application that forecasts daily the Earth ionosphere. The experiments run FIFO, SJF, EASY-backfilling and Fattened-backfilling algorithms for 3 different scenarios, and we could observe FIFO presented the worse performance. A better performance for SJF was achieved in some metrics. For favorable and unfavorable scenarios EASY-backfilling and Fattened-backfilling were able to achieve similar performance of SJF.

The results with the scientific application were not significantly different from each other because there were only 2 types of jobs, and the number of jobs to schedule was relatively low. So, the algorithms were also submitted to a controlled scenario using NAS benchmark using more diverse types of jobs and workload. In that case, Fattened-backfilling algorithm was superior in advancing smaller jobs, while EASY-backfilling algorithm presented better results in execution time when compared to the other algorithm. We observed a decrease in performance for the more simple algorithms evaluated (FIFO and SJF).

Although the experiments were performed in a real cluster, it is possible some weaknesses of job scheduling algorithms have not emerged, probably because the cluster size was limited. We suggest, as future work, to apply the job scheduling and resource management in different scientific computing environments, as well as to evaluate them using larger clusters.

Acknowledgments. The author Fernando Emilio Puntel thanks Coordenaçño de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) for the financial support in this research.

References

1. Arndt, O., Freisleben, B., Kielmann, T., Thilo, F.: A comparative study of online scheduling algorithms for networks of workstations. Cluster Comput. **3**(2), 95–112 (2000)
2. Bailey, D.H.: NAS parallel benchmarks. In: Padua, D. (ed.) Encyclopedia of Parallel Computing, pp. 1254–1259. Springer, Boston (2011). https://doi.org/10.1007/978-0-387-09766-4
3. Bailey, D.H., et al.: The NAS parallel benchmarks. Int. J. Supercomput. Appl. **5**(3), 63–73 (1991)

4. Natural Resources Canada: Monthly averages of solar 10.7 cm flux. http://www.spaceweather.gc.ca/solarflux/sx-5-mavg-en. Acessado em 20 de julho de 2018
5. Casalicchio, E., Perciballi, V.: Measuring docker performance: what a mess!!! In: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion, ICPE 2017 Companion, pp. 11–16. ACM, New York (2017)
6. Ciliendo, E., Kunimasa, T., Braswell, B.: Linux performance and tuning guidelines. IBM, International Technical Support Organization (2007). https://lenovopress.com/redp4285.pdf
7. Feitelson, D.G., Rudolph, L., Schwiegelshohn, U., Sevcik, K.C., Wong, P.: Theory and practice in parallel job scheduling. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1997. LNCS, vol. 1291, pp. 1–34. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63574-2_14
8. Georgiou, Y.. et al.: A scheduler-level incentive mechanism for energy efficiency in HPC. In: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 617–626. IEEE (2015). https://ieeexplore.ieee.org/abstract/document/7152527
9. Georgiou, Y., Glesser, D., Trystram, D.: Adaptive resource and job management for limited power consumption. In: 2015 IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW), pp. 863–870. IEEE (2015). https://ieeexplore.ieee.org/abstract/document/7284402
10. Gibbons, R.: A historical application profiler for use by parallel schedulers. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1997. LNCS, vol. 1291, pp. 58–77. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63574-2_16
11. Gómez-Martín, C., Vega-Rodríguez, M.A., González-Sánchez, J.-L.: Fattened backfilling: an improved strategy for job scheduling in parallel systems. J. Parallel Distrib. Comput. **97**, 69–77 (2016)
12. Mao, W., Kincaid, R.K.: A look-ahead heuristic for scheduling jobs with release dates on a single machine. Comput. Oper. Res. **21**(10), 1041–1050 (1994)
13. Mu'alem, A.W., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Trans. Parallel Distrib. Syst. **12**(6), 529–543 (2001)
14. Petry, A., et al.: First results of operational ionospheric dynamics prediction for the Brazilian space weather program. Adv. Space Res. **54**(1), 22–36 (2014)
15. Skovira, J., Chan, W., Zhou, H., Lifka, D.: The EASY—LoadLeveler API project. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1996. LNCS, vol. 1162, pp. 41–47. Springer, Heidelberg (1996). https://doi.org/10.1007/BFb0022286
16. Subbulakshmi, T., Manjaly, J.S.: A comparison study and performance evaluation of schedulers in hadoop yarn. In: 2017 2nd International Conference on Communication and Electronics Systems (ICCES), pp. 78–83, October 2017
17. Talby, D., Feitelson, D.G.: Supporting priorities and improving utilization of the IBM SP scheduler using slack-based backfilling. In: 13th International and 10th Symposium on Parallel and Distributed Processing Parallel Processing, 1999. 1999 IPPS/SPDP. Proceedings, pp. 513–517. IEEE (1999)
18. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Backfilling using system-generated predictions rather than user runtime estimates. IEEE Trans. Parallel Distrib. Syst. **18**(6), 789–803 (2007)
19. Vasupongayya, S., Chiang, S.-H.: On job fairness in non-preemptive parallel job scheduling. In: IASTED PDCS, pp. 100–105. Citeseer (2005)

20. Wong, A.K., Goscinski, A.M.: The impact of under-estimated length of jobs on easy-backfill scheduling. In: 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing, pp. 343–350. IEEE (2008). https://ieeexplore.ieee.org/abstract/document/4457142
21. Yoo, A.B., Jette, M.A., Grondona, M.: SLURM: simple Linux utility for resource management. In: Feitelson, D., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2003. LNCS, vol. 2862, pp. 44–60. Springer, Heidelberg (2003). https://doi.org/10.1007/10968987_3
22. Zhou, X., Chen, H., Wang, K., Lang, M., Raicu, I.: Exploring distributed resource allocation techniques in the SLURM job management system. Technical report Illinois Institute of Technology, Department of Computer Science (2013)