The Journal of Supercomputing (2024) 80:13738–13753
https://doi.org/10.1007/s11227-024-05988-z

Enhancing heterogeneous cluster efficiency through node-centric scheduling

Esteban Stafford[1] · Jose Luis Bosque[1]

Accepted: 9 February 2024 / Published online: 11 March 2024

Abstract
This article delves into the critical realm of modern computer cluster management. It focuses on the effect that the increasing heterogeneity of the clusters has on the workload managers. The proposed schedulers consider node properties instead of job properties to make decisions, which is something not currently done by mainstream scheduling algorithms. In order to increase the knowledge in this topic, this paper proposes two novel algorithms whose main task is to choose the best compute nodes to schedule the incoming jobs. To this effect, they exclusively take into account the properties of the nodes, instead of the common trend of considering the properties of the jobs. The experimental results show that these algorithms outperform well-known heuristic algorithms found in the literature.

Keywords Heterogeneous clusters · Scheduler · Energy efficiency · Performance

1 Introduction

Modern society relies heavily on the computing capacity of thousands of data centres spread around the world. These centralised hubs of computational power are the backbone of numerous industries, ranging from cloud computing and e-commerce to scientific research and artificial intelligence. Commonly, they house computer clusters of different sizes, that execute massive amounts of computing jobs. The huge growth experimented by these systems in the last years has greatly increased the energy consumption of the IT industry in general. Therefore,

Stafford Esteban and Bosque Jose Luis have contributed equally to this work.

✉ Esteban Stafford
esteban.stafford@unican.es

Jose Luis Bosque
joseluis.bosque@unican.es

1 Department of Computer Engineering and electronics, Universidad de Cantabria, Avda. los castros, s/n, 39005 Santander, Spain

efficiently exploiting these resources not only translates to improved performance, but also reduced energy consumption and enhanced environmental sustainability.

Workload managers and schedulers are a very important part of the adequate management and operation of large high-performance computing (HPC) clusters [1]. This software is in charge of receiving jobs from different users and assigning them the necessary resources for their correct execution. This allocation generally seeks to optimise a specific objective, such as application performance, cluster utilisation or, more recently, energy consumption or efficiency.

It is well known that the resource allocation problem is NP-Complete [2]. Several optimisation algorithms have been proposed that focus on optimising objectives in the short term, but they are very time-consuming and not very practical [3]. For this reason, current workload managers, such as Slurm [4], PBS [5], Moab/Torque [6] or Cobalt [7], use simple heuristic algorithms to solve the problem, such as first come first served (FCFS) or backfill [8]. These algorithms focus on selecting the most appropriate job to schedule at each moment in time. However, they do not take into account the details of the computation nodes and simply choose the first one that is available, without any specific criteria. Even new proposals of much more sophisticated scheduling algorithms, such as DRAS [9] or RLScheluder [10], completely ignore the characteristics of the system, considering the cluster as a set of identical computational resources.

These approaches oversimplify modern clusters that are becoming increasingly heterogeneous [11]. That is, they have a large number of resources with different properties, like number of cores, frequency or energy consumption [12]. In these heterogeneous systems, the choice of the node on which to execute a particular job can have a very strong impact on the objective pursued by the scheduler. Hence, the choice of the node becomes as important, or even more, than the choice of the job to be scheduled. Therefore, in this paper we hypothesise that for heterogeneous clusters, it is possible to define simple heuristic node-selecting algorithms that outperform traditional algorithms that focus on job selection instead.

In this paper, we propose a study of two simple greedy scheduling algorithms, which put their intelligence in the selection of the compute node. The algorithms consider that the nodes are composed solely of general purpose CPUs, without any accelerators. Since they only focus on node selection, the job selection policy is in all cases FCFS. The algorithms adhere to the shared-memory paradigm, allocating jobs entirely to a single node. The first proposal, called HighGFlops, always selects the best performing node from those available at each scheduling step. The second one, called LowPower, always selects the node that consumes the least power, from those available at any given time. The experimental results presented in this article confirm our hypothesis, since the algorithms achieve reductions of 11.5% in the total execution time over the baseline algorithms and a 17% in energy consumption.

The main contributions of this article are the following:

- To purpose and validate two heuristic and greedy scheduler algorithms, based on the selection of the compute nodes, instead of the jobs to be scheduled.

- To analyse the impact that the cluster heterogeneity has in the scheduling, for different objectives, such as performance, energy consumption and energy efficiency.

This article is structured as follows. Section 2 presents the proposed algorithms. Section 3 describes the experimental methodology. Section 4 shows the experimental results. Finally, Sect. 6 presents the conclusions and future work.

2 Algorithm proposal

In the context of heterogeneous cluster computing environments, it is pertinent to highlight the stark contrast between the variability of the characteristics of the jobs and the relatively constant properties of the nodes that compose a heterogeneous cluster [13].

In the literature, there are plenty of examples of studies pondering on the wide array of attributes and parameters that describe a computational task [1, 3, 8–10, 14–19]. These can include factors such as the computational intensity of a task, its memory requirements, parallelisation potential or data access patterns. These characteristics can vary significantly when users execute different applications, change the size or complexity of their data-sets, or simply shift from a parameter search with serial codes to a massively parallel simulation. In fact, nowadays there is a large proliferation of big-data and artificial intelligence (AI) applications that have very different requirements and memory access patterns compared to traditional scientific or engineering workloads. All in all, it can be argued that endeavouring to accommodate the ever-changing complexity of job characteristics presents a formidable challenge. Nonetheless, there are numerous scheduling algorithms that attempt to order jobs in a way that gives a performance improvement.

Modern clusters have evolved into heterogeneous environments, where the diversity of hardware components and configurations presents new scheduling opportunities. These are often disregarded by traditional scheduling algorithms that were primarily designed for homogeneous clusters and consider clusters as a set of identical computing resources. Since the hardware of clusters usually represents a huge economic investment, the properties of their nodes can be considered almost constant [20]. These properties encompass a range of hardware specifications, such as processor architectures, clock speeds, memory sizes and storage capacities. Additionally, they can include information about the network connectivity, power consumption profiles and hardware accelerators present in each node.

This study presents a novel paradigm for scheduling algorithms that are explicitly designed to address the inherent heterogeneity of nodes within the cluster. Unlike conventional scheduling methodologies, the proposed node-centric approach refrains from any form of job prioritisation or sorting. Instead, it can be considered as a heterogeneous variant of the FCFS scheduling policy. The primary objective of these algorithms is to make informed decisions regarding the selection of nodes for the execution of incoming jobs, leveraging the relative stability of node properties.

Each one of the presented algorithms is tailored to improve different metrics and tackle different scenarios. The HighGFlops scheduling algorithm is designed to improve computational performance. In contrast, the LowPower scheduling algorithm seeks to improve energy consumption. These algorithms collectively offer versatile solutions to accommodate the diverse demands of modern heterogeneous computer clusters.

Both algorithms work with a job queue ordered by submission time, encompassing a diverse collection of application types. The heterogeneous cluster that will execute the jobs is composed of compute nodes, characterised by unique properties such as the number of cores, clock frequency, instructions per cycle (IPC) or power consumption. Consequently, the allocation of applications to specific nodes yields varying response times and energy consumption levels based on the inherent attributes of the selected nodes.

These algorithms are subject to the constraint that each job must be assigned to a single node, and each task or thread in the job must be executed exclusively by a single core. This limitation is derived from the simulator used in the evaluation, that does not feature a network communication model and therefore is unable to provide accurate results for distributed applications.

The power consumption of a node is modelled in three fundamental states. Firstly, when all cores are in an idle state the node consumes a minimal amount of power. Secondly, a static power consumption level rises when at least one core is actively running, indicating that common circuitry in the node is supporting some sort of computation. Finally, each core individually incurs its specific energy consumption while actively running, contributing to the overall power consumption of the system.

2.1 HighGFlops

The HighGFlops scheduling algorithm is based on choosing the *fastest* node available in the cluster that has enough cores to execute the incoming job. Determining this is possible considering a simplification of the processor performance equation (PPE) [21]. This equation relates the execution time of a task t to its number of instructions of the application I, the clock frequency of the CPU f and the instructions per cycle (IPC) factor with the following expression:

$$t = \frac{I}{f \cdot IPC}$$

The number of instructions is determined by the application and does not change based on scheduling decisions, but estimating IPC proves challenging due to its dependence on the specific interplay between the executed applications and the underlying CPU architecture. Therefore, the HighGFlops algorithm simplifies this model by considering the IPC is constant for all the jobs and nodes in the cluster. Then, to improve the execution time t it tries to allocate incoming jobs to the cores with the highest clock frequency f available.

The consideration of IPC as a constant factor across all jobs and nodes represents a pragmatic approach by the HighGFlops algorithm. While this simplification may not capture the nuanced variations in IPC that arise from different applications and CPU architectures, it allows for more straightforward scheduling decisions. By prioritising cores with higher clock frequencies, HighGFlops aims to expedite task execution, particularly benefiting compute-intensive workloads. The algorithm's adherence to memory-sharing paradigms and the constraint that all allocated cores must belong to a single node further refines its approach, aligning resource allocation strategies with the practicalities of cluster computing. This algorithmic approach, while simplifying certain aspects of the performance model, endeavours to enhance overall computational performance in a manner favourable to real-world cluster scenarios.

This algorithm is particularly effective when dealing with CPU-bound applications characterised by a limited IPC range. In such cases, the algorithm optimally assigns jobs to nodes with the highest clock frequencies while leaving slower nodes idle, efficiently utilising computational resources. However, it is important to notice that the power consumption of nodes is intimately related to their clock frequency, causing faster nodes to consume more power. Also, since the energy is defined as the product of time and power:

$$e = t \cdot P$$

The energy consumption is affected by these two factors. The HighGFlops algorithm reduces the execution time of the jobs by using the faster nodes, which has a positive impact on the energy consumption. On the other hand, the use of these nodes will increase the power consumption and, therefore, can negatively affect the energy consumption of the cluster. The result of these contrasting effects will need to undergo evaluation via experimentation.

2.2 LowPower

Concerns regarding power consumption have become a crucial issue in the context of computer clusters. Furthermore, the overall power usage of a cluster is closely linked to the decisions made by the scheduling algorithm. The LowPower scheduling algorithm has been devised to address these concerns. It is known that manufacturers typically label their products with a maximum power specification, effectively condensing the complex power consumption of their products into a single value. The operation of the LowPower algorithm is as follows: incoming jobs are systematically assigned to nodes that have enough idle cores and the lowest power consumption level.

This algorithm minimises the power consumption of the nodes, which is expected to have a positive impact in the energy consumption of the cluster. However, the nodes with the least power consumption are commonly less performant. Then, the execution time of the jobs will be higher, consequently increasing the energy consumption. Therefore, it is likely that the LowPower algorithm will not have the best performance, but it is reasonable to believe that it will have a good behaviour in terms of energy consumption. This is something that will be evaluated experimentally as it depends on the balance between the energy saving and the increase of the execution time.

Figure 1 shows an example of how the presented algorithms operate. It can be seen that HighGFlops allocates the first task (0) to the fastest node (3GHz), the second (1) to the next node in speed (2GHz) and so forth. Since power consumption is proportional to frequency in this example, the LowPower algorithm does the opposite. Note that the first (0) task goes to the slowest node (1GHz) in this case.

3 Methodology

It can be argued that the evaluation of these algorithms could be done in a real cluster, but this would be a very expensive and time-consuming task. Therefore, it has been carried out in IRMaSim, a simulator designed to study the behaviour of scheduling algorithms in computer clusters [22]. Its main requirement is that it must model the behaviour of clusters in the most realistic way but with very high speedups. IRMaSim allows a more precise cluster simulation, compared to similar solutions, by supporting finer grain modelling of node architectures, and giving the possibility of configuring heterogeneous clusters, where nodes do not have to be identical. It simulates contention from shared resources, such as memory, and features a more suitable power consumption model for multi-core architectures. Both contention and power models were validated against a real cluster.

To evaluate the new proposals of this article, these are compared to other scheduling algorithms, MinMin, MaxMin and Duplex [23]. These were chosen because, unlike other classic algorithms like FCFS or shortest job first (SJF), they consider the heterogeneity of the cluster. The main idea behind these algorithms is a non-greedy scheduling that aims to allocate tasks to compute resources in a way that minimises their response time. The MinMin does this with the jobs sorted in growing request time order, and conversely the MaxMin sorts the jobs in diminishing request time. This behaviour is illustrated in Fig. 1. Note how the

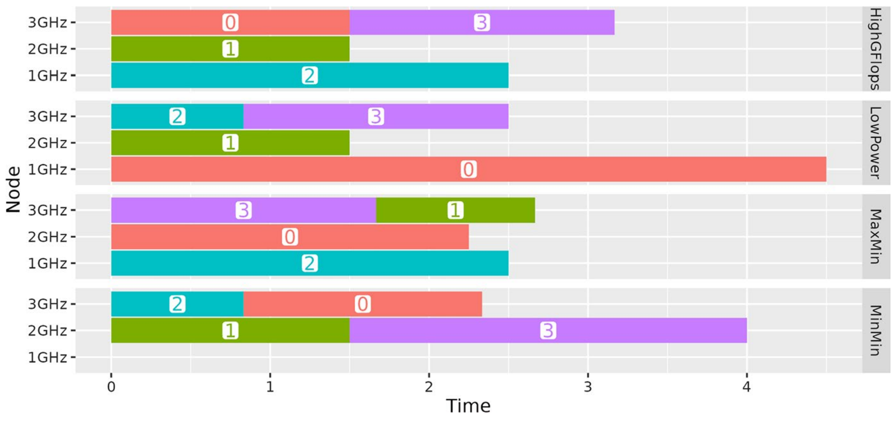

Fig. 1 Example scheduling of different algorithms

MinMin algorithm allocates the shortest jobs (1,2) to the fastest nodes first and it does not use the slowest node (1GHz) since the long jobs would greatly increase the total execution time. In contrast, MaxMin allocates the longest jobs (0,3) to the fastest nodes and is able to allocate one of the sort jobs (2) to the slow node (1GHz) without compromising the total execution time. The Duplex algorithm, actually compares the scheduling of both MinMin and MaxMin and selects the one with the shortest global completion time. In order to estimate the response time of each task, these algorithms consider the requested time of the job and the clock frequency of the nodes, thus becoming aware of the heterogeneity of the cluster. As depending on the clock frequency of the nodes, the jobs are going to have different execution time estimations.

The experiments presented are based on a simulated cluster that has 16 nodes as described in Table 1. Every row describes a group of nodes, starting with the node count and continuing with its properties. The Min column gives the power consumption when all the cores are idle, the Static column expresses the power drawn by the node when at least one core is allocated, and the Dynamic column is the power needed to feed one running core.

The workloads are based on the KIT-FH2-2016-1 workload trace. This job log contains the activity of the Karlsruhe Institute of Technology ForHLR II System over 582 days scheduling more than 114 thousand jobs, with an average 199 jobs per day. The average duration of the jobs is 4.9h and the maximum number of cores per job has been artificially reduced to 64 to emulate the shared-memory jobs. The average number of cores per job is 24.6.

Various metrics have been used to evaluate the effectiveness of the different schedulers.

- Makespan is the total time needed to execute all the jobs in the trace.
- Average waiting time considers the time each job is queued before it executes.
- Slowdown is the ratio of the response time of each job to its baseline execution time.
- Energy measures the consumption of the cluster during the makespan.
- Energy delay product (EDP) is an efficiency metric calculated as the product of the makespan and the energy [24].

Table 1 Composition of the cluster

Count	Cores	Freq.	Power		
			Min	Static	Dynamic
4	64	1GHz	0.4W	3.9W	0.4W
4	64	2GHz	0.4W	15.6W	1.5W
4	64	3GHz	0.4W	35.1W	3.3W
4	64	4GHz	0.4W	62.4W	5.9W

Note that the slowdown is defined slightly different than in other articles. In homogeneous clusters, the slowdown is calculated with the execution time of the job.

$$Slowdown = \frac{t_w + t_e}{t_e}$$

where t_w is the waiting time and t_e is the execution time. Since in heterogeneous clusters t_e varies depending on the node that executes the job, this definition is no longer accurate. Therefore, in this article it is replaced in the denominator by the baseline execution time t_b. This is defined as the estimated execution time of the job running alone in the slowest node of the cluster.

$$Slowdown = \frac{t_w + t_e}{t_b}$$

This expression preserves the spirit of the slowdown as a metric of user dissatisfaction, but it includes the beneficial effect of running on the faster nodes of a cluster. If a value of 1 in the traditional definition means that there was no waiting time, in the above expression the result can be lower than 1 if the job was executed in one of the faster nodes. Values lower than 1 can happen in the presence of waiting time if the executing node is fast enough.

4 Evaluation

To better assess the effectiveness of the schedulers, the trace has been divided into weeks and these have been simulated independently. The results of these weekly simulations are presented in the box-and-whisker graphs shown in Fig. 2. There is one graph for each metric, where the x-axis covers all the schedulers and the y-axis represents the value of each metric. The boxes show the 25 and 75 percentile, the line in the box indicates the median and the whiskers represent extreme values.

From a performance perspective, the relevant metrics are makespan, waiting time and slowdown. Regarding the waiting time results, the proposed algorithms are usually capable of launching jobs as soon as they are submitted. Both have fairly low waiting times. The LowPower median is 75% lower than the baseline and for the HighGFlops it is 80% lower. In both cases, Q3 is close to zero, meaning that in 25% of the weeks there is almost no waiting time.

Regarding the slowdown results, the proposed algorithms show an excellent improvement over the baseline ones. In detail, the median slowdown for LowPower is 75% less than the baseline algorithms. This can be attributed to the good waiting times it showed. But with HighGFlops, the results are much better because it systematically chooses the fastest nodes available, further reducing the response time over the baseline execution time. It can be seen that the median slowdown of HighGFlops is 94% lower than the baselines.

As each of the algorithms favour a type of node, they have a different impact in the makespan and energy metrics. The HighGFlops sends jobs to the fastest

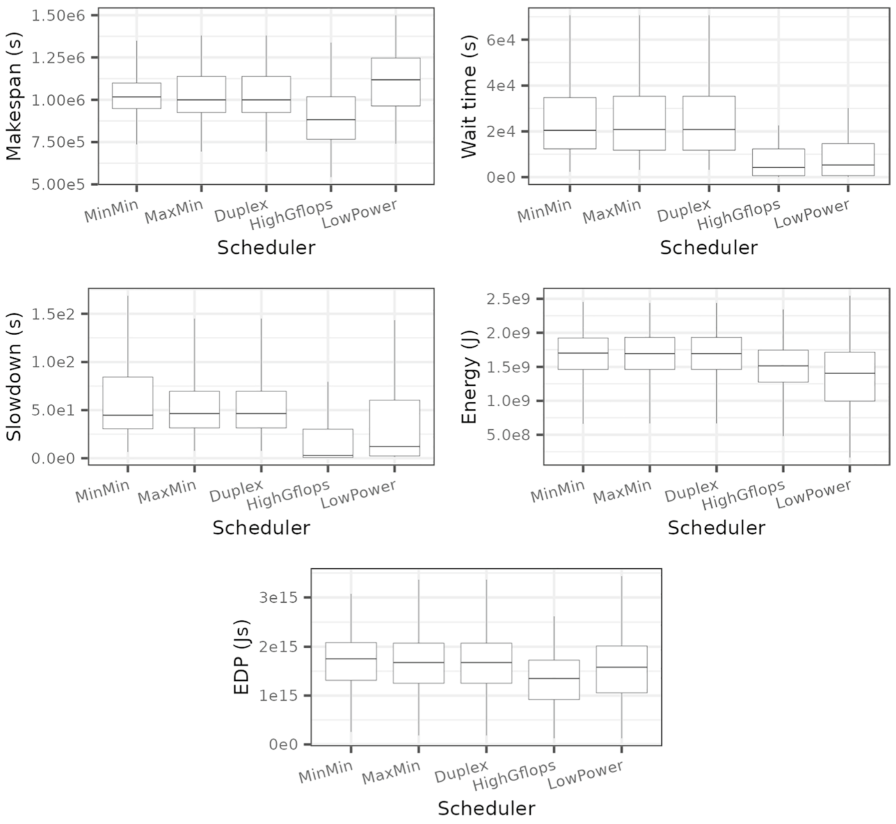

Fig. 2 Performance and energy results

nodes if they are available. As a consequence, it is able to give very good results in the total execution time of the traces, with a 11.5% reduction in the median of the makespan. On the contrary, LowPower chooses the nodes with the least possible power consumption, which are also the slowest. This has a bad effect in the makespan results, in fact its median is 11% worse than those of the baseline schedulers.

Attending to the energy results, the HighGFlops scheduler has good results, in fact it has a 10% better consumption than the baseline. But the LowPower scheduler is the best with a median improvement of 17%. These results help understand the impact that the power consumption of the nodes and their execution speeds have in the overall energy consumption of the cluster. In this case, the LowPower scheduler benefits from using slower nodes that consume less power, even if they take longer to execute.

Finally, the EDP metric is a combination of the makespan and the energy, reflecting the energy efficiency of the cluster. The LowPower has some improvement over the baselines, with a median that is 10% lower. It is noteworthy that the good energy consumption of this algorithm overcomes its lack in performance.

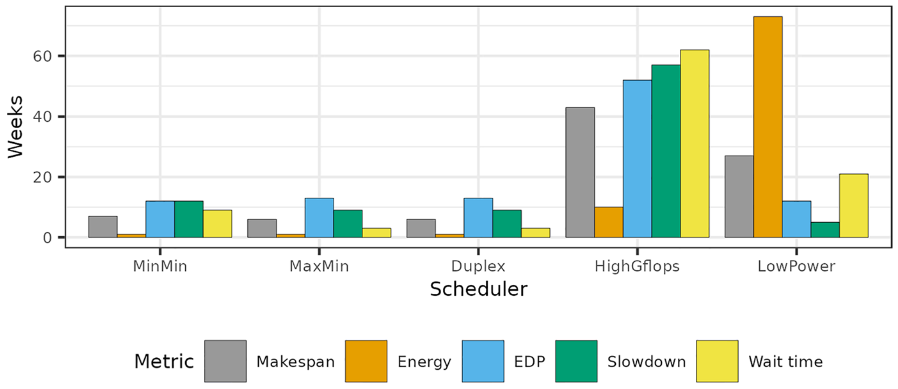

Fig. 3 Number of weeks each algorithm is the best

But the HighGFlops has much better results with a 33% lower median, which comes as a consequence that its good results in makespan and energy.

The previous results reduce the weekly results to their statistical values. In order to better understand the behaviour of the algorithms, Fig. 3 shows the number of weeks each algorithm is the best in each metric. The x-axis represents the algorithms and the y-axis the number of weeks. It is noteworthy, that these counts consider ties for the best result, in which case both are considered best. This different perspective confirms the previous evaluation and reveals the good results of the proposed algorithms compared to the baseline. Indeed, they show that HighGFlops is the most frequent winner in makespan and EDP. The opposite happens in energy consumption where the LowPower is most often the best scheduler.

To better understand the behaviour of the algorithms regarding their performance and energy consumption, a representative week has been selected, for which detailed graphs of various metrics are shown. To avoid clutter, only the Duplex baseline algorithm appears in the graphs. It was chosen for being the most sophisticated among the baselines.

This analysis is summarised in two graphs, Fig. 4 shows the amount of energy consumed by the cluster under the management of three different schedulers. The

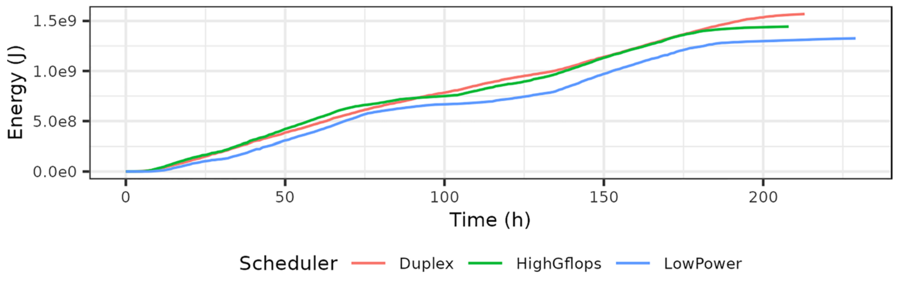

Fig. 4 Time evolution of energy consumption

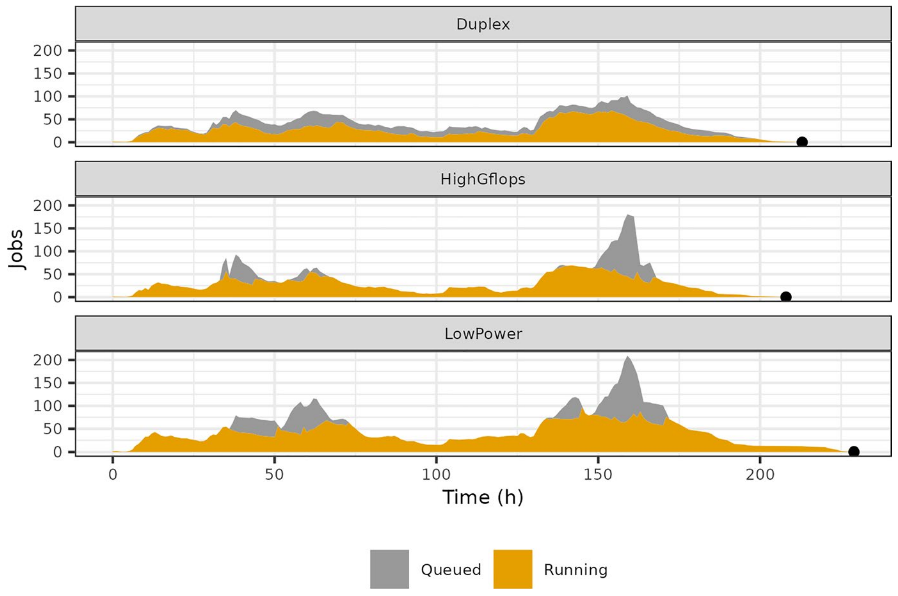

Fig. 5 Running and queued job evolution over time

maximum x value of each of the curves indicates the makespan. Figure 5 represents the evolution of the number of running and queued jobs at each moment in time. Also, the makespan is indicated by a black dot on the x-axis. The experiment considers the full execution of all the tasks submitted during this time. Thus, makespan values are usually well above the weekly 168 h.

The bad results of the Duplex scheduler are apparent in both Figures. It can be seen in Fig. 4 that for most of the execution the energy consumption is similar to that of HighGFlops, but after a point, due to an inability to schedule jobs at the submission rate, it ends up taking more time and consuming more energy.

The Duplex algorithm relies on an precise execution time prediction to achieve a tight packing of the jobs. However, the inherent inaccuracy of the user-requested time, which references the slowest node, coupled with how this time is scaled on faster nodes, leads to subpar decisions by the algorithm.

The Figures also illustrate that by employing the lowest-power nodes, the LowPower scheduler achieves the lowest energy consumption, albeit at the cost of a longer makespan compared to HighGFlops. On the contrary, the HighGFlops consumes more energy, but finalises all the job sooner than LowPower. The difference in energy consumption and performance of these two schedulers, that concentrate the jobs in certain nodes, is possible because there are times at which the cluster is not fully occupied. This means that LowPower does not have to resort to the fast and power-hungry nodes and, conversely, the HighGFlops can avoid using the slow nodes of the cluster. Note how the slope of the energy graph, which is the power, is similar in the three algorithms when there are queued jobs, but diverges when the machine is less busy.

5 Related work

The introduction of this work stated that scheduling in HPC systems is performed by a software called workload manager, being the most known Slurm [4], PBS [5], Moab/Torque [6] or Cobalt [7]. These workload managers implement simple heuristic algorithms, which are based on selecting jobs waiting in a queue sorted according to different job properties, and assigning them to the first available node in the cluster. These range from simple one-parameter heuristic priority functions such as FCFS to complex metrics combining multiple parameters like WFP, UNICEP [25] or F1 [26].

Another frequently employed technique is *resource reservation* or *backfilling* [8]. This is a look-ahead approach that creates a job schedule, usually based on a traditional algorithm like FCFS, but it uses the user-provided execution time estimation to fill idle gaps that may appear in the schedule with lower priority jobs. EASY backfilling is a more aggressive refinement that allows the lower priority jobs to delay the original schedule [13]. Backfilling provides good results on production systems, but relies heavily on job execution time estimates being realistic, and its performance is greatly affected by inaccuracies [27]. In any case, these algorithms are centred on the ordering and selection of jobs, but they always select any free node of the cluster, regarding the cluster as a homogeneous set of nodes.

Another relevant aspect influencing cluster performance is the choice between two scheduling paradigms: *list scheduling* or *pack scheduling* [28]. In list scheduling, pending jobs are initially organised in a queue before being dispatched sequentially, while in pack scheduling, jobs are partitioned into packs. Jobs within each pack are scheduled concurrently, and the next pack cannot start executing until all the jobs in the previous pack have finished running. While this distinction affects the organisation of jobs in the queue, the system is once again modelled as a homogeneous cluster.

Tackling the heterogeneous nature of current computer clusters has been attempted in previous articles. In addition to the baselines described above, MinMin, MaxMin and Duplex, there are others like A* [29]. A* is a complex heuristic based on a tree structure. In the context of job scheduling, the search tree represents possible sequences of job schedules. The algorithm explores this tree while using heuristics to guide its search towards the most promising leafs, ultimately reaching the optimal solution [30].

This article primarily emphasises heuristic algorithms due to their lightweight nature. However, recently machine learning techniques have been applied to address the job-scheduling problem. For instance, RLScheduler [10] uses deep reinforcement learning (DRL), feeding a neural network-based agent a vector that contains the attributes of all the eligible jobs. The agent selects one job that is allocated to the first free node. RLSchert [19] is also based on DRL and remaining runtime prediction. It estimates the state of a cluster by means of a dynamic job remaining runtime predictor that employs a recurrent neural network to encode time series information. Deep reinforcement agent for scheduling

(DRAS) [31] takes advantage of the resource reservation technique, combining backfilling with a DRL on a hierarchical neural network for decision-making. All these algorithms are focused in selecting jobs and model the cluster as a set of homogeneous nodes, without consider heterogeneity. Some efforts have been made to extend the RLScheduler to heterogeneous clusters [32], but this work focused on performance optimisation, not energy consumption or efficiency.

6 Conclusion

This article introduces two novel heuristic algorithms for job scheduling in heterogeneous clusters. These algorithms are primarily focused on improving various metrics, mainly the makespan and the energy consumption. To do this, they take into account the heterogeneity of the components of the cluster. The HighGFlops consider the different clock frequencies of the cluster nodes and, by prioritising faster nodes, it attempts to reduce the execution time of the jobs. In contrast, the LowPower scheduler is designed to reduce the total energy consumption of the cluster by sending jobs to nodes that consume less power whenever possible. These algorithms were compared to a set of schedulers known as MinMin, MaxMin and Duplex, that are also sensitive to the properties of the nodes. But the presented schedulers have the ability of reducing waiting times, contributing to favourable outcomes, and do not rely on user-provided estimations of execution time. In fact, the evaluation results indicate that these heuristics can outperform traditional schedulers in multiple metrics, with at least one algorithm emerging as the winner in each metric. Based on these results, it can be said that the LowPower scheduler is the one indicated to reduce the energy consumption, whereas the HighGFlops should be the one to choose if a improvement of the makespan or the energy efficiency is required.

Future developments related to this research are the following. First, the IRMaSim simulator could be improved to enhance the realism of traces containing message passing interface (MPI) jobs. This entails building a model into the simulator to predict the performance penalties derived from network communication among tasks. Additionally, IRMaSim could also incorporate dynamic voltage and frequency scaling (DVFS) modelling, aiming to better simulate the energy consumption of modern processors. Modern heterogeneous clusters include accelerators of different kinds. The presented algorithms could be improved to successfully tackle these cases. Seeing that none of the presented algorithms is the best for all the metrics, it could be worthwhile to explore a method for intelligently selecting the most suitable scheduler for each job. This approach would enable the utilisation of the strengths of both algorithms, leading to enhanced overall system performance and efficiency. Finally, by integrating these algorithms into real-world workload managers, tangible improvements in system performance and efficiency could be achieved.

Acknowledgements The KIT ForHLR II workload log (https://www.cs.huji.ac.il/labs/parallel/workload/l_kit_fh2/index.html) was graciously provided by Mehmet Soysal.

Author contributions Both authors contributed equally to the manuscript.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. This work has been supported by the Spanish Science and Technology Commission under contracts PID2019-105660RB-C22 and PID2022-136454NB-C21, the Ministerio de Ciencia e Innovación; Proyectos de Transición Ecológica y Digital 2021 under grant TED2021-131176B-I00 and the European HiPEAC Network of Excellence.

Data availability The trace data used in this study is available at https://www.cs.huji.ac.il/labs/parallel/workload/l_kit_fh2/index.html

Code availability The simulator used in this study, IRMaSim, is available at https://github.com/irmasim/IRMaSim.

Declarations

Ethical approval Not applicable.

References

1. Allcock W, Rich P, Fan Y, Lan Z (2018) Experience and practice of batch scheduling on leadership supercomputers at argonne. In: Job scheduling strategies for parallel processing: 21st international workshop, JSSPP 2017, Orlando, FL, USA, June 2, 2017, Revised Selected Papers 21, pp 1–24. Springer
2. Ullman JD (1975) Np-complete scheduling problems. J Comput Syst Sci 10(3):384–393
3. Fan Y, Lan Z, Rich P, Allcock WE, Papka ME, Austin B, Paul D (2019) Scheduling beyond cpus for hpc. In: Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing. HPDC '19, pp 97–108. Association for Computing Machinery, New York, NY, USA
4. Yoo AB, Jette MA, Grondona M (2003) Slurm: Simple linux utility for resource management. In: Feitelson D, Rudolph L, Schwiegelshohn U (eds) Job Scheduling Strategies for Parallel Processing. Springer, Berlin, Heidelberg, pp 44–60
5. Feng H, Misra V, Rubenstein D (2007) Pbs: a unified priority-based scheduler. In: Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp 203–214
6. Declerck TM, Sakrejda I (2013) External torque/moab on an xc30 and fairshare. Technical report, NERSC
7. Desai N (2005) Cobalt: an open source platform for hpc system software research. In: Edinburgh BG/L System Software Workshop, pp 803–820
8. Leonenkov S, Zhumatiy S (2015) Introducing new backfill-based scheduler for slurm resource manager. Procedia Comput Sci 66:661–669 (**4th International Young Scientist Conference on Computational Science**)

9. Fan Y, Li B, Favorite D, Singh N, Childers T, Rich P, Allcock W, Papka ME, Lan Z (2022) Dras: Deep reinforcement learning for cluster scheduling in high performance computing. IEEE Trans Parallel Distrib Syst 33(12):4903–4917
10. Zhang D, Dai D, He Y, Bao FS, Xie B (2020) RLscheduler: an automated HPC batch job scheduler using reinforcement learning. In: SC20: Int. Conf. for High Performance Computing, Networking, Storage and Analysis, pp 1–15. IEEE
11. Stafford E, Bosque JL (2020) Improving utilization of heterogeneous clusters. J Supercomput 76(11):8787–8800
12. Bosque JL, Toharia P, Robles OD, Pastor L (2013) A load index and load balancing algorithm for heterogeneous clusters. J Supercomput 65(3):1104–1113
13. Mu'alem AW, Feitelson DG (2001) Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. IEEE Trans Parallel Distrib Syst 12(6):529–543
14. Mao H, Schwarzkopf M, Venkatakrishnan SB, Meng Z, Alizadeh M (2019) Learning scheduling algorithms for data processing clusters. In: Proceedings of the ACM Special Interest Group on Data Communication. SIGCOMM '19, pp 270–288
15. Maroulis S, Zacheilas N, Kalogeraki V (2019) A holistic energy-efficient real-time scheduler for mixed stream and batch processing workloads. IEEE Trans Parallel Distrib Syst 30(12):2624–2635
16. Shamsa E, Kanduri A, Liljeberg P, Rahmani AM (2021) Concurrent application bias scheduling for energy efficiency of heterogeneous multi-core platforms. IEEE Trans Comput 71(4):743–755
17. Fan Y (2021) Job scheduling in high performance computing
18. Dupont B, Mejri N, Da Costa G (2020) Energy-aware scheduling of malleable hpc applications using a particle swarm optimised greedy algorithm. Sustain Comput: Inf Syst 28:100447
19. Wang Q, Zhang H, Qu C, Shen Y, Liu X, Li J (2021) Rlschert: an hpc job scheduler using deep reinforcement learning and remaining time prediction. Appl Sci 11(20):9448
20. Nozal R, Perez B, Bosque JL, Beivide R (2019) Load balancing in a heterogeneous world: Cpu-xeon phi co-execution of data-parallel kernels. J Supercomput 75(3):1123–1136
21. Hennessy JL, Patterson DA (2017) Computer Architecture, Sixth Edition: A Quantitative Approach, 6th edn. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA
22. Herrera A, Ibáñez M, Stafford E, Bosque J (2021) A simulator for intelligent workload managers in heterogeneous clusters. In: 2021 IEEE/ACM 21st Int. Sym. on Cluster, Cloud and Internet Computing (CCGrid), pp 196–205
23. Maheswaran M, Ali S, Siegal HJ, Hensgen D, Freund RF (1999) Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In: Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99), pp 30–44
24. Castillo E, Alvarez L, Moreto M, Casas M, Vallejo E, Bosque JL, Beivide R, Valero M (2018) Architectural support for task dependence management with flexible software scheduling. In: 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp 283–295
25. Tang W, Lan Z, Desai N, Buettner D (2009) Fault-aware, utility-based job scheduling on blue, gene/p systems. In: IEEE International Conference on Cluster Computing and Workshops, pp 1–10
26. Tang W, Lan Z, Desai N, Buettner D (2009) Fault-aware, utility-based job scheduling on blue, gene/p systems. In: 2009 IEEE International Conference on Cluster Computing and Workshops, pp 1–10
27. Tsafrir D, Etsion Y, Feitelson DG (2007) Backfilling using system-generated predictions rather than user runtime estimates. IEEE Trans Parallel Distrib Syst 18(6):789–803
28. Sun H, Elghazi R, Gainaru A, Aupy G, Raghavan P (2018) Scheduling parallel tasks under multiple resources: List scheduling vs. pack scheduling. In: 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp 194–203
29. Braun TD, Siegel HJ, Beck N, Bölöni LL, Maheswaran M, Reuther AI, Robertson JP, Theys MD, Yao B, Hensgen D, Freund RF (2001) A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J Parallel Distribut Comput 61(6):810–837
30. Shahul S, Zaki A, Sinnen O (2010) Scheduling task graphs optimally with a*. J Supercomput 51(3):310–322
31. Fan Y, Li B, Favorite D, Singh N, Childers T, Rich P, Allcock W, Papka ME, Lan Z (2022) Dras: Deep reinforcement learning for cluster scheduling in high performance computing. IEEE Trans Parallel Distrib Syst 33(12):4903–4917
32. Fomperosa J, Ibáñez M, Stafford E, Bosque JL (2022) Task scheduler for heterogeneous data centres based on deep reinforcement learning. In: 14th International Conference Parallel Processing and

Applied Mathematics PPAM 2022, Gdansk, Poland, September 2022. Lecture Notes in Computer Science, vol. 13826, pp 237–248