Impact of a dynamic Allocation Policy for Resource and Job Management Systems in deadline-oriented Scenarios

Barry Linnert[1], Cesar Augusto F. De Rose[2], Hans-Ulrich Heiss[1]

[1]Fakultät für Elektrotechnik und Informatik – Technische Universität Berlin

[2]Faculdade de Informática – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)

{barry.linnert,hans-ulrich.heiss}@tu-berlin.de, cesar.derose@pucrs.br

Abstract. *As High Performance Computing (HPC) becomes a tool used in many different workflows, Quality of Service (QoS) becomes increasingly important. In many cases, this includes the reliable execution of an HPC job and the generation of the results by a certain deadline. The Resource and Job Management System (RJMS or simply RMS) is responsible for receiving the job requests and executing the jobs with a deadline-oriented policy to support the workflows. In this paper, we evaluate how well static resource management policies cope with deadline constrained HPC jobs, and explore two variations of a dynamic policy in this context. Our preliminary results clearly show that a dynamic policy is needed to meet the requirements of a modern deadline-oriented RMS scenario.*

1. Introduction

In recent years, High Performance Computing (HPC) has become a much-used tool in many application domains, such as science and development, as well as business and industry. Many new applications in this broad range of fields are being developed to take advantage of the ever-increasing number of nodes that make up HPC systems, also known as supercomputers. This challenge of implementing new approaches to parallel programming that make meaningful use of the enormous compute power is accompanied by the need to embed these applications into some high-level workflows.

This use of supercomputers, as well as the integration into specific workflows, is familiar from science, where deadline-driven workflows are common, but is also important for many other application scenarios in development and industry. For example, when developing a new type of airplane engine, data from current engine versions are incorporated alongside objectives such as reducing noise and fuel consumption. Simulations implemented as finite element method (FEM) therefore have to provide the results for optimizations of the engine blades or combustion chamber in time for the engineering team so that the prototype and subsequently the final product can be built as previously planned. In this context the resource management system (RMS), which is sometimes also referred to as resource and job management system (RJMS), is responsible for allocating HPC resources efficiently to incoming jobs to improve throughput and also has to guarantee job deadlines in order to provide the quality of service (QoS) requested by the users [Fan 2021][Fan et al. 2022]. Therefore, in a QoS-oriented environment, the RMS has to decide *if* a job request submitted to the HPC system can be accepted, taking into account the deadline of the job, *when* and *where* on the machine the job is to be executed [Le Hai et al. 2020]. Only when all questions are answered positively can QoS be

guaranteed and the job can be scheduled and afterwards executed on the HPC system. This is the only way to ensure that the deadline specified by the workflow in which the compute job is embedded is met. This is a challenging task because the final execution time of a job depends on several factors that are hard to predict, such as resource demand and availability throughout the execution, their specific communication patterns, and how well these patterns match the machine's memory hierarchy and network topology. The latter is especially true for any system that implements a multi-level network topology, and is even more important in the context of the fastest supercomputers, such as the former number 1, the Titan supercomputer, or Fugaku the current number 2 in the current TOP 500 list [Strohmaier et al.]. When users are asked to give the execution times for their jobs, they usually fail to give good estimates because they have no running history or technical knowledge to make predictions, or because they want to trick the system to get ahead of other users.

Specifically, our contributions to the state of the art in this paper are as follows:

- We examined how state of the art RMSs with static allocation policies would cope with deadline-oriented scenarios.
- We apply a dynamic resource allocation policy to an RMS and evaluate how a contiguous and a non-contiguous variant compares to the static policy.
- Based on the analysis of results obtained through an extensive number of simulations under different workload conditions, we demonstrate the advantages of dynamic policies in deadline-oriented scenarios over the state of the art.

2. State of the Art in HPC Resource Management

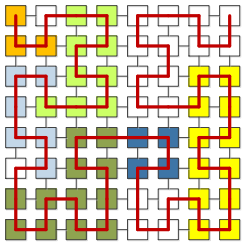


Figure 1. Example of a mapping with Hilbert curve

Current RMSs implement static allocation policies assigning a set of compute nodes or CPU cores for the entire execution of a job at start time. The parallel program then assigns the processes of the job to the nodes of CPU cores at runtime, often with assistance from middleware and the RMS. This typically results in over-provisioning for programs with dynamic runtime behavior in order to guarantee that additional resources needed during execution are available, and reduces resource utilization. When the job is finished, or when the specified end time of the job is reached, the CPU core or compute nodes are released and marked as free to be assigned to the next job or set of jobs. Like the SLURM workload manager [Yoo et al. 2003], a queuing-based resource management system widely used in the area of HPC that implements a batch job-based static strategy for handling job requests. In fact, it is the most popular RMS for the HPC systems summarized in the TOP500 list. The system can also be extended to support advance reservation [Becker 2021] in order to provide a higher level of QoS needed to support deadline-oriented resource management.

Since the shape of the partition to which the job is assigned is important to minimize the communication overhead for many parallel applications and HPC systems, especially those implementing a mesh/grid or torus topology such as the previously mentioned Titan and Fugaku, the SLURM workload manager aims to provide compact, contiguous partitions to the jobs.

Therefore, SLURM uses a shaping component that makes use of a Hilbert curve approach. When using the Hilbert curve, the neighborhood relationship, which is the basis for creating compact partition shapes, is transferred from two- or more-dimensional grid or torus topology to a one-dimensional line of nodes (see the red line in Figure 1).

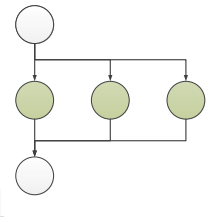


Figure 2. Runtime behavior of a Monte Carlo-like program

With this Hilbert curve, the SLURM workload manager uses a first fit approach (by following the red line) in order to find a partition with sufficient number of nodes or CPU cores for the job request and still form compact partitions that should help to reduce the communication overhead for the job. The partition formed using the Hilbert curve is then assigned to the job and can be used by the parallel program over the entire runtime, scheduled by the RMS. The SLURM workload manager uses the Hilbert curve approach primarily to support three- or more-dimensional grid or torus topologies. In order to investigate the approach, it has also been implemented for two-dimensional grid topologies.

Nevertheless, the RMS today – in addition to providing resource for the job – is more and more responsible for providing a certain level of quality of service (QoS). When it comes to supporting higher-level workflows, this level of QoS is, as mentioned earlier, related to the reliable execution of the parallel program that is submitted to the HPC system as a job. In this case, reliable execution is often embedded in some form of contract – the Service Level Agreement (SLA). In a SLA the service provider – the provider of the HPC resource – regularly guarantees the agreed quality of service and would pay some kind of fee if the level of quality of service is not achieved. Therefore, in these environments, the execution of the job must be completed before the deadline specified in the SLA. Hence, resource management systems for HPC systems have to be examined to determine whether they currently meet these QoS requirements, and if not, new approaches need to be developed. Since there are different types of parallel programs running on the supercomputers, the examination has to be performed with respect to these different types of applications.

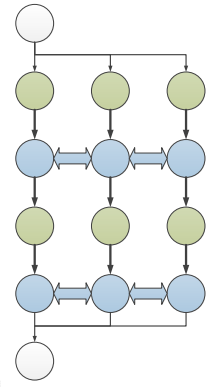


Figure 3. Runtime behavior of a BSP-based program

3. Parallel Applications and Load of HPC Jobs

Because the range of applications domains that make use of HPC is enormous, there are different approaches to creating parallel applications and, consequently, different types of resource requirements for these programs [Frank 2022]. In many cases, the parallel programs use message passing to distribute and collect data to be processed by the various processes that make up the running program – the compute job. In the era of multi-core node architectures, the use of shared memory provided by these nodes also comes into play. Since the characteristics of the different programs resulting from the different programming approaches play an important role in providing efficient support for these parallel applications, a closer look at the runtime behavior of the programs is needed.

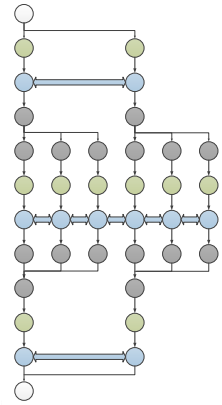


Figure 4. Runtime behavior of a program using OpenMP

Parallel programs, designed as so-called Monte Carlo applications, consist of a fixed number of processes processing randomly generated data (see Fig. 2, where light gray circles symbolize start and end nodes of the program; light green circles symbolize the execution of instructions (computations) of the processes). The processes of the job run independently and do not exchange intermediate data. The only utilization of network resources occurs when input data is distributed to the processes and the results are received in order to be evaluated and output is provided. This type of application is used primarily when statistical evidence is needed, such as for discrete event simulations that simulate technical entities like HPC or Grid environments. However, this type of application is also used for implementing various approximation approaches and for simulating physical processes. Nowadays, HPC systems are also used to train machine learning (ML) models. These types of applications also exhibit this type of runtime behavior.

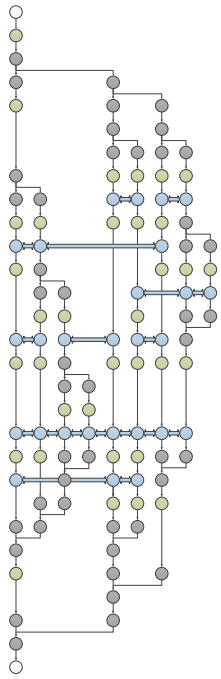


Figure 5. Dynamic runtime behavior of a program using MPI-2

Another type of parallel programs is also based on a fixed number of processes, but the processes do not run independently (see Fig. 3). Instead, they exchange data during processing in order to provide intermediate data or results to other processes and to obtain new input data for the next step of computation itself. One of the most commonly used models for building such a parallel program is the Bulk-Synchronous Parallel model (BSP) [Valiant 1990].

The BSP approach is often used to build applications that simulate sections of the real world that are geometrically divided into individual parts. These parts are interconnected and represent an area of the real world with a certain state which depends on the development of the states of the surrounding areas. An example of such an application of the finite element method (FEM) is the simulation of the flow behavior around an airplane engine blade. Other examples include the simulation of natural systems such as weather or the Earth's climate. In recent years, the introduction of multi-core CPUs on the one hand and of the development of new features for the Message Passing Interface (MPI) leading to the new standard MPI-2, as well as the increased use of OpenMP in combination with message passing approaches on the other hand, have led to the emergence of new types of parallel programs with new runtime behavior (see Fig. 4). In order to utilize multiple cores of a CPU or the processors of a node, OpenMP is used to launch additional processes. The creation of new and additional processes can be based on the design of the application, so that the number of processes to be started is known before the job is started, or the start of additional processes can be controlled by the pro-

gram at runtime. This creation of additional processes based on runtime decisions, e.g., about the amount of work to be processed, has already been introduced in approaches such as the bag-of-task or the manager-worker approach, but is increasingly used in applications implementing finite elements methods. Simulations of particles and molecules also benefit from this approach, which leads to dynamic runtime behavior of the programs. The creation of additional processes can be used in a more flexible way when using the new features provided by the MPI-2 standard. With MPI-2, new processes can be created on different nodes and the start of the additional processes or threads is not limited to the local node, already used by some MPI processes of the job (see Fig. 5) [Perez et al. 2017][Aguilar Mena et al. 2022]. The models presented here are only examples of very small programs. In real scenarios and during evaluation, the number of tasks reaches up to billions, especially for programs with dynamic runtime behavior.

4. Dynamic Allocation of HPC Resources

For parallel programs implementing a more static runtime behavior, such as Monte Carlo-like applications or BSP programs (types 0 and 1), the static allocation as provided by the current RMS, like the SLURM workload manager, should be well suited to support these types of runtime behavior in a deadline-oriented workflow scenario. However, for the applications with dynamic runtime behavior (types 2 and 3), it can be assumed that this approach is not appropriate because the number of compute resources changes during runtime [Álvarez et al. 2022][Li et al. 2023]. Therefore, a more dynamic way of resource assignment can be beneficial. This means the resources required by the running job are assigned at the time the resource is needed and is released immediately after use.

In order to implement such dynamic allocation of HPC resources and support decentralized and distributed resource management, the leak approach was introduced in [Heiss 1994], [De Rose 1998], and discussed in detail in [De Rose et al. 2007]. In this work we have adapted it to work with RMSs.

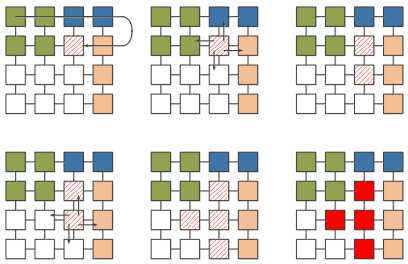


Figure 6. Search for first nodes using the Leak approach

In the leak approach, the parallel program resembles a liquid that is dropped into a – usually partially occupied – container. Due to its properties, the job occupies a free part of the container or displaces another job until an equilibrium is reached. Since parallel programs – especially those with dynamic runtime behavior – change their properties, such as the number of processes running in parallel, the equilibrium and thus the number of resources used by the different jobs can change over time. This breath-like expansion of the number of CPU cores or nodes used by the parallel program ideally fits into a phase of reduced node utilization of other jobs. Thus, the jobs should complement each other in terms of resource utilization. Thus, when a job is submitted to the local resource management system, the job is accepted – no further check is performed.

The job is then started at the earliest possible start time. To start the first process of the parallel program, an empty compute node or CPU core is searched for. The distri-

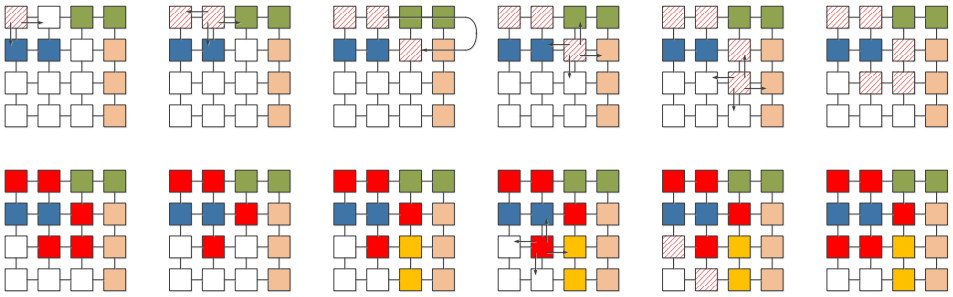


Figure 7. Search for nodes using the Non-contiguous Leak approach: after the initial partition is created in the first row (red nodes), nodes are released in the second row during execution, and new nodes are assigned when new processes are created

bution of the different jobs over the entire HPC system is supported by the use of different nodes from which the search is started. If the entry node is occupied and therefore unable to accommodate the new process, a next node is examined. The search for a first CPU core or node for the new job is performed sequentially for all nodes or core of the same row for grid or torus topologies, as shown in Figure 6. Thus, the search sequence for a first node for the starting job resembles a snake. In case no node or CPU core is available for the job to be started, the job is rejected.

After starting the program some other processes can be created. In connection with the creation of the process there is a search for a CPU core or nodes on which the process is to be started. The search for the core or node is performed starting from the node or CPU core to which the creating process is assigned. First, all of the neighboring nodes or cores are checked. If no core or node of these adjacent nodes or cores is available, all other nodes or cores used by the parallel program are used to check the neighbors of these nodes or CPU cores. In this way, a compact partition of nodes should be created for the processes of the running program. In case no free node or CPU core from the set of neighboring nodes is available to satisfy the current request, the search for other nodes can be performed like the search for a start node as described before (see Fig. 7). This results in *non-contiguous* partitions for the job. If only neighboring nodes or CPU cores are considered to be included into the partition, this is referred to as *contiguous* leak approach. When the process using a node or CPU core is finished, this node or core is released and the then free node can be used by other processes of the same or other running parallel programs.

5. Evaluation

Based on the observations presented earlier and with respect to the different types of parallel programs, the examples of static and dynamic resource allocation were examined. In order to evaluate the different approaches of resource management systems in terms of their suitability to provide the required quality of service and the ability to ensure the completion of the job at a given time, we used the ApplOXSim simulation environment presented in [Schneider and Linnert 2014] and used in [Linnert et al. 2014] to study the impact of data structures and static allocation policies.

In order to evaluate the dynamic allocation policies, the two variations of the leak approach were implemented to be used in this simulation environment (contiguous and non-contiguous). The search for a new CPU core is performed when a process invokes a fork system call and the CPU core is released when a join call is performed by the process. Furthermore, the simulator is able to support different configurations of HPC machines. The configurations used for the evaluation presented in this paper are based on the performance values of the supercomputer HLRN-II, which was in operation at the Zuse Institute Berlin (ZIB). Here, the system is used with 512 CPU cores on 128 compute nodes connected by a grid topology. To provide a reasonable load for the RMS to handle, the workload generator by Feitelson [Feitelson et al. 2014] was used and the resulting traces were extended to include the type of the runtime behavior of the parallel programs as well as the earliest start time and deadline for the job. Since deadlines are defined by the users and the higher-level workflow in which the job is embedded, it is reasonable to assume that the deadline leaves some leeway for the management system. This slack time usually depends on the importance and the size of the job, as the user would want to reduce the risk of canceling the job if the results are important. The workload generator can be configured to provide different load levels in the form of arrival rates. In addition to the recommended and common load, which is referred to as normal load and is determined with an interarrival factor of 1500, two other load situations were examined. The very heavy load (interarrival factor of 1), where a new job request is submitted to the RMS every second on average. Between the normal and very heavy load, a heavy load situation was investigated with an interarrival factor of 150.

The traces provide 1000 job requests over a simulation time of about 65 days for the heavy and very heavy load configurations, and about 74 days for the normal load traces. By using these configuration values combined with Feitelson's workload generator, based on the extensive empirical research in the field [Feitelson et al. 2014], our results can be compared with other approaches from related work. The running parallel programs were derived from the different runtime behavior types and adapted to the specifications as given in the job description of the individual trace file.

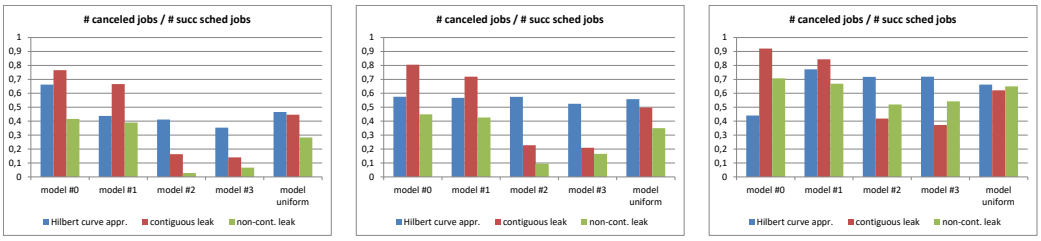
In addition to the simulation of traces with jobs following the same runtime behavior type (models 0 to 3), a mixture of the types was used according to a uniform distribution of the behavior types (model uniform). For each configuration – application type, communication behavior, load – 320 different traces were generated and used for simulation in order to obtain significant results for the performance of the different approaches. In total, about 30,000 different simulation runs were performed to obtain the results resented in this paper. [CURTA]

5.1. Job Canceling Rate

The comparison of the two different approaches – static and dynamic resource assignment – in terms of the number of successfully scheduled jobs suggests that not all jobs can be completed successfully – especially for dynamic allocation. The results for the leak approach confirm this assumption for both the contiguous and the non-contiguous version of the leak approach (see Fig. 8(a)) even under normal load. However, a significant number of canceled jobs is also observed for the Hilbert curve-based approach used by SLURM.

As expected, the share of canceled jobs increases with load in the dynamic approaches (see Fig. 8(c)). In the static approach, this is due to the congestion situation in

the network, and in the dynamic approaches it is due to the case where no free node or CPU core is found at the time a new process is created. The latter is even more relevant if the number of nodes to be examined is reduced, since only nodes in the neighborhood of the existing partition are considered, as is the case with the contiguous leak approach. Nevertheless, the results for the very heavy load show that the increased communication overhead associated with a more scattered assignment, due to the need to use more distant nodes in the case of non-contiguous leak, and a higher load on the network links themselves results in worse performance for the non-contiguous leak compared to the contiguous leak for the job with dynamic runtime behavior.



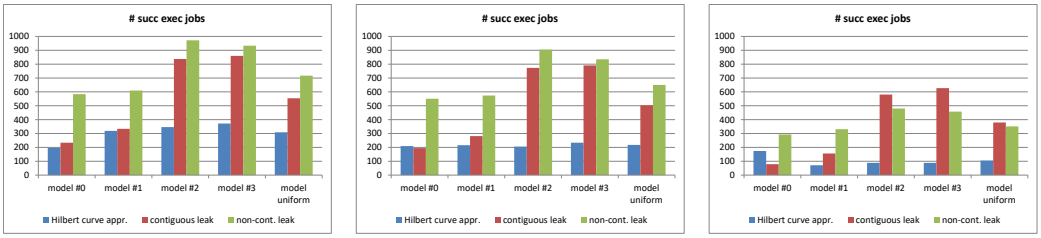
(a) Normal load (b) Heavy load (c) Very heavy load

Figure 8. The canceling rate depends on the combination of load, behavior model, and approach, but in almost all configurations, the non-contiguous leak approach (green) outperforms the Hilbert curve-based approach (blue).

While for the dynamic assignment approaches the share of canceled jobs is related to the number of canceled jobs, the normalized number of canceled jobs also increases for the SLURM system – except for the Monte Carlo-like programs (runtime behavior type 0). Since communication is concentrated at the start and end of the runtime, communication overhead is not as important to the runtime as it is for other types of runtime behavior. Moreover, the higher load – which comes with a reduction in interarrival time – in case of static assignments reduces the advantage of smaller jobs. If all job requests arrive at the same time and the decision to schedule some of the jobs based on the number of available nodes or CPU cores, all jobs have an equal chance of being selected, and the preference for smaller jobs that fit into some holes in an already existing schedule is negligible. Since the larger jobs are associated with higher slack time, these jobs have a higher probability to get finished before the specified deadline. However, due to the better support of the runtime behavior of the jobs, especially those with dynamic runtime behavior, the dynamic allocation performs better than the static approach based on the Hilbert curve when it comes to the share of canceled jobs.

5.2. Overall Performance

The unexpected high number of canceled jobs in combination with the rejection of jobs due to the allocation of static partitions provided by the Hilbert curve-based SLURM approach leads to reduced performance in terms of the number of successfully executed jobs. This holds true for the jobs with dynamic runtime behavior, where only half the number of jobs can be completed before the deadline compared to the leak approaches under normal load (see Fig. 9(a). The results are even more evident for the other load situations (see Fig. 9(b) and Fig. 9(c)). However, also for the jobs, implementing a static runtime behavior using the static partition approach is beneficial only for Monte Carlo-like applications



(a) Normal load (b) Heavy load (c) Very heavy load

Figure 9. Due to the high acceptance rate and the canceling rate, the leak approaches (red and green) perform better than the Hilbert curve-based approach (blue) in terms of successfully executed jobs, especially for the jobs with dynamic runtime behavior.

at very heavy load compared to the contiguous leak approach, while the non-contiguous leak performs better even for this type of applications. Thus, the characteristics of the static partition approach, which comes with a guarantee to the job that there will always be enough compute nodes or CPU cores available to start the processes, and the shape that reduces communication overhead, do not compensate for the specific requirements associated with more sophisticated new programming approaches implementing a dynamic runtime behavior. The fundamental results observed for the programs with asynchronous communication generally hold for the jobs with synchronous communication patterns, although there are some increases for certain configurations. For the SLURM RMS and the contiguous leak approach, the synchronization of the processes through synchronous communication leads to a decrease in the number of canceled jobs and thus to an increase in the number of successfully executed programs compared to the job with asynchronous communication. This can be seen particularly well when looking at the results for the jobs implementing a BSP-like runtime behavior (type 1) in combination with the contiguous leak approach. In this case, the occasional mapping of the child process to the node of the parent process, since no free node is available in the neighborhood of the partition, leads to a reduction of the communication overhead and can at least partially compensate for the overload situation on the respective node.

6. Related Work

A lot of research is being done in the field of HPC management [Reed et al. 2023][Shilpika et al. 2022][Alam et al. 2022]. Besides the development of novel applications [Qiu et al. 2022][Nesi et al. 2022] [Aguilar Mena et al. 2022] and the improvement of RMS, such as SLURM, research continues in many ways on scheduling of HPC jobs [Nichols et al. 2022] [Ueter et al. 2022] [Zrigui et al. 2022] or the mapping of the processes as part of the programs to the processors or nodes [Frank 2022][Li et al. 2022][Zrigui et al. 2022]. In addition, the requirement that applications must have generated the result by a certain deadline is also increasingly taken into account [Le Hai et al. 2020] [Fan 2021]. All these aspects and requirements have to be considered in order to develop a reliable solution for the next generation of HPC resource management systems. Nevertheless, to the best of our knowledge, there is no other work that has explored and analyzed the impact of a dynamic RMS strategy in the context of deadline-oriented execution scenarios – and its impact on scheduling,

mapping, application performance and throughput - with respect to the different types of runtime behavior.

7. Conclusion and Outlook

In this paper, we evaluate how well RMS static resource management policies cope with deadline-constrained HPC jobs and explore two variations of a dynamic policy in such scenarios. Deadline enforcement is crucial in environments where a higher level of quality of service (QoS) is implemented through the use of service level agreements (SLA) in order to support advanced workflows.

We perform extensive simulation experiments using SLURM, the state of the art in resource management systems, as a static baseline and compare its results with a dynamic policy adapted to this context.

Our preliminary results clearly show the impact of the runtime behavior of the parallel applications running on these HPC systems on the results, as was also suggested in [Fan 2021], and that a static policy is not able to meet the requirements of a modern deadline-oriented RMS scenario, especially in terms of resource utilization and the number of jobs that are successfully executed as agreed in the SLA.

Rather, the results for the dynamic assignment approaches – the contiguous approach and even more so the non-contiguous leak approach – show the need to support dynamic runtime behavior of the jobs in the form of providing additional compute nodes or CPU cores at the time a new process is created. The neighborhood relationship between the nodes of the partition is important in reducing the communication overhead associated with the message passing of the parallel applications, but has less impact than the timely provisioning of additional compute power. Since this effect may be of minor importance in queueing-based resource management systems where the runtime of the job can be extended without major impact (except for optimizations implementing backfilling [Frachtenberg et al. 2003]) and the reduction in utilization associated with static partitions can be accepted, any aspect that threatens the deadline has to be omitted in the area of SLA-based RMS.

However, the results for the dynamic partitioning – the leak approaches – also show that the resource requirements of parallel programs with the same runtime behavior type can match in such a way that one program can use the currently freed resources that were previously used by another program. This effect can greatly increase the number of jobs executed on the HPC system. Though the mixture of parallel applications with different runtime behavior types can reduce this effect.

Therefore, our preliminary results clearly show that RMS should adopt dynamic policies, but more detailed information about the – future – runtime behavior of the parallel program should be incorporated in the scheduling and mapping decisions performed by the RMS. Future work will address this feature and pursue a resource management approach that is capable of providing a reliable service for the execution of parallel programs with deadlines by taking into account the runtime behavior and resource requirements of the HPC jobs in order to support SLAs.

References

Aguilar Mena, J., Shaaban, O., Beltran, V., Carpenter, P., Ayguade, E., and Labarta Mancho, J. (2022). Ompss-2@ cluster: Distributed memory execution of nested openmp-style tasks. In *Euro-Par 2022: Parallel Processing: 28th International Conference on Parallel and Distributed Computing, Glasgow, UK, August 22–26, 2022, Proceedings*, pages 319–334. Springer.

Alam, S. R., Bartolome, J., Carpene, M., Happonen, K., s LaFoucriere, J.-C., and Pleiter, D. (2022). Fenix: A Pan-European Federation of Supercomputing and Cloud e-Infrastructure Services. *Communications of the ACM*, 65(4).

Álvarez, D., Sala, K., and Beltran, V. (2022). nos-v: Co-executing hpc applications using system-wide task scheduling. *arXiv preprint arXiv:2204.10768*.

Becker, R. P. (2021). Entwurf und Implementierung eines Plugins für SLURM zum planungsbasierten Scheduling. Bachelor's Thesis, Freie Universität Berlin.

CURTA. Curta: A General-purpose High-Performance Computer at ZEDAT, Freie Universität Berlin. `https://doi.org/10.17169/refubium-26754` (visited May 19, 2021).

De Rose, C. A. (1998). *Verteilte Prozessorverwaltung in Multirechnersystemen*. PhD thesis, Universität Karlsruhe (Technische Hochschule).

De Rose, C. A., Heiss, H.-U., and Linnert, B. (2007). Distributed dynamic processor allocation for multicomputers. *Parallel Computing*, 33(3):145–158.

Fan, Y. (2021). Job scheduling in high performance computing. *Horizons in Computer Science Research*, 18.

Fan, Y., Lan, Z., Rich, P., Allcock, W., and Papka, M. E. (2022). Hybrid workload scheduling on hpc systems. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 470–480. IEEE.

Feitelson, D. G., Tsafrir, D., and Krakov, D. (2014). Experience with using the parallel workloads archive. *Journal of Parallel and Distributed Computing*, 74(10):2967–2982.

Frachtenberg, E., Feitelson, D. G., Fernandez, J., and Petrini, F. (2003). Parallel job scheduling under dynamic workloads. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 208–227. Springer.

Frank, A. (2022). *Reducing resource waste in HPC through co-allocation, custom checkpoints, and lower false failure prediction rates*. PhD thesis, Johannes Gutenberg-Universität Mainz.

Heiss, H.-U. (1994). *Prozessorzuteilung in Parallelrechnern*. BI-Wiss.-Verlag.

Le Hai, T. H., Trung, K. P., and Thoai, N. (2020). A working time deadline-based backfilling scheduling solution. In *2020 International Conference on Advanced Computing and Applications (ACOMP)*, pages 63–70. IEEE.

Li, B., Fan, Y., Dearing, M., Lan, Z., Rich, P., Allcock, W., and Papka, M. (2022). Mrsch: Multi-resource scheduling for hpc. In *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 47–57. IEEE.

Li, J., Michelogiannakis, G., Cook, B., Cooray, D., and Chen, Y. (2023). Analyzing resource utilization in an hpc system: A case study of nersc perlmutter. *arXiv preprint arXiv:2301.05145*.

Linnert, B., Schneider, J., and Burchard, L.-O. (2014). Mapping algorithms optimizing the overall Manhattan distance for pre-occupied cluster computers in SLA-based Grid environments. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 132–140. IEEE.

Nesi, L. L., Schnorr, L. M., and Legrand, A. (2022). Multi-phase task-based HPC applications: Quickly learning how to run fast. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 357–367. IEEE.

Nichols, D., Marathe, A., Shoga, K., Gamblin, T., and Bhatele, A. h. (2022). Resource utilization aware job scheduling to mitigate performance variability. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 335–345. IEEE.

Perez, J. M., Beltran, V., Labarta, J., and Ayguadé, E. (2017). Improving the integration of task nesting and dependencies in openmp. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 809–818. IEEE.

Qiu, H., Xu, C., Li, D., Wang, H., Li, J., and Wang, Z. (2022). Parallelizing and balancing coupled DSMC/PIC for large-scale particle simulations. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 390–401. IEEE.

Reed, D., Gannon, D., and Dongarra, J. (2023). Hpc forecast: Cloudy and uncertain. *Communications of the ACM*, 66(2):82–90.

Schneider, J. and Linnert, B. (2014). List-based data structures for efficient management of advance reservations. *International Journal of Parallel Programming*, 42(1):77–93.

Shilpika, S., Lusch, B., Emani, M., Simini, F., Vishwanath, V., Papka, M. E., and Ma, K.-L. (2022). Toward an in-depth analysis of multifidelity high performance computing systems. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 716–725. IEEE.

Strohmaier, E., Dongarra, J., Simon, H., Meuer, M., and Meuer, H. Top500 list. `https://www.top500.org/` (visited April 25, 2021).

Ueter, N., Günzel, M., von der Brüggen, G., and Chen, J.-J. (2022). Parallel path progression DAG scheduling. *arXiv preprint arXiv:2208.11830*.

Valiant, L. G. (1990). A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111.

Yoo, A. B., Jette, M. A., and Grondona, M. (2003). Slurm: Simple Linux utility for resource management. In *Workshop on job scheduling strategies for parallel processing*, pages 44–60. Springer.

Zrigui, S., de Camargo, R. Y., Legrand, A., and Trystram, D. (2022). Improving the performance of batch schedulers using online job runtime classification. *Journal of Parallel and Distributed Computing*, 164:83–95.