SiL: An Approach for Adjusting Applications to Heterogeneous Systems Under Perturbations

Ali Mohammed and Florina M. Ciorba
Department of Mathematics and Computer Science
University of Basel, Switzerland

May 28, 2022

Contents

Abstract

Scientific applications consist of large and computationally-intensive loops. Dynamic loop scheduling (DLS) techniques are used to load balance the execution of such applications. Load imbalance can be caused by variations in loop iteration execution times due to problem, algorithmic, or systemic characteristics (also, perturbations). The following question motivates this work: *"Given an application, a high-performance computing (HPC) system, and both their characteristics and interplay, which DLS technique will achieve improved performance under unpredictable perturbations?"* Existing work only considers perturbations caused by variations in the HPC system delivered computational speeds. However, perturbations in available network bandwidth or latency are inevitable on production HPC systems. *Simulator in the loop* (SiL) is introduced, herein, as a new control-theoretic inspired approach to dynamically select DLS techniques that improve the performance of applications on heterogeneous HPC systems under perturbations. The present work examines the performance of six applications on a heterogeneous system under *all above system perturbations*. The SiL *proof of concept* is evaluated using simulation. The performance results confirm the initial hypothesis that *no single DLS technique can deliver best performance in all scenarios*, while the SiL-based DLS selection delivered improved application performance in most experiments.

Keywords. Performance Load balancing Loop scheduling Heterogeneous computing systems Perturbations Simulation

1 Introduction

Scientific applications are often characterized by large and computationally-intensive parallel loops. The performance of these applications on high-performance computing (HPC) systems may degrade due to load imbalance caused by problem, algorithmic, or systemic characteristics. Application (problem or algorithmic) characteristics include the irregularity of the number of computations per loop iterations due to conditional statements, where systemic characteristics include variations in delivered computational speed of processing elements (PEs), available network bandwidth or latency. Such variations are referred to as perturbations, and can also be caused by other applications or processes that share the same resources, or a temporary system fault or malfunction. Dynamic loop scheduling (DLS) is a widely-used approach for improving the execution of parallel applications using self-scheduling, that is *dynamic* assignment of the loop iterations to free and requesting processing elements. A wide range of DLS techniques exists, and can be divided into *nonadaptive* and *adaptive* techniques. The nonadaptive DLS techniques account for the variability in loop iterations execution times due to application characteristics. They do not account for irregular system characteristics that are known only during execution. The nonadaptive DLS techniques include *self-scheduling* (SS), *fixed size chunking* (FSC) [14], *guided self-scheduling* (GSS) [20], *factoring* (FAC) [12], and *weighted factoring* (WF) [11]. The adaptive DLS techniques account for irregular system characteristics by adapting the amount of assigned work per PE request (chunk size) according to the application performance measured during execution. Adaptive DLS techniques include *adaptive weighted factoring* (AWF) [3], its variants *batch* (AWF-B), *chunk* (AWF-C), *batch-like* (AWF-D), *chunk-like* (AWF-E) [7], and *adaptive factoring* (AF) [2].

An *a priori* selection of the most appropriate DLS technique for a given application and system is challenging, given the various sources of load imbalance and the different load balancing properties of the DLS techniques. This observation raises the following question and motivates the present work: *"Given an application, an HPC system, and both their characteristics and interplay, which DLS technique will achieve improved performance under unpredictable perturbations?"* Earlier work studied the flexibility of DLS (robustness to reduced delivered computational speed) [22] and the selection of robust DLS using machine learning [23] with the SimGrid (SG) [8] simulation toolkit. The selection of DLS techniques for synthetic time-stepping scientific applications using reinforcement learning [4] was also studied using SG. The aforementioned existing work focuses on one source of perturbations (variation in delivered computing speed) in time-stepping applications to learn from previous steps. That approach may not be applicable to applications without time-steps, nor would it be feasible in a highly variable execution environment. Scheduling solutions using static optimizations, e.g., using evolutionary and genetic algorithms, can not dynamically adapt to the perturbations encountered during execution. Modern HPC systems are often heterogeneous production systems typically shared by many users. Therefore, perturbations in the available network bandwidth and latency in such systems are unavoidable.

In the present work, in an effort to select the most appropriate DLS for a given application and system, the performance of a scientific application (PSIA [10]) and five synthetic applications using nonadaptive and adaptive DLS techniques is studied on a heterogeneous HPC system, in the presence of perturbations in computing speed, network bandwidth, and network latency. The amount of operations in each loop iteration of the five synthetic applications is assumed to follow five different probability distributions, namely: constant, uniform, normal, exponential, and gamma probability distributions. The present work makes the following contributions: (1) Proposes a novel *simulator in the loop* (SiL) approach for dynamically selecting a DLS technique during execution, based on the application characteristics and the present (monitored or predicted) state of the computing system; (2) Provides insights on the resilience of the DLS techniques to perturbations; and (3) Confirms the initial hypothesis that no single DLS ensures the best performance in all execution scenarios considered; The SiL performance is evaluated for the selected applications in simulation using SG.

This work is structured as follows. Section 2 contains a brief review of the selected DLS techniques, the SG simulation toolkit, as well as of the work related to the performance of scheduling scientific applications with DLS in the presence of perturbations. The proposed SiL approach for selecting a DLS technique in the presence of perturbations is discussed in Section 3. The experimental design and setup, and the performance of the proposed approach are described and discussed in Section 4. The work concludes and outlines potential future work in Section 5.

2 Background and Related Work

Loop scheduling. The aim of loop scheduling is to achieve a balanced load execution among the parallel PEs with minimum scheduling overhead. Loop scheduling can be divided into *static* and *dynamic*. In static loop scheduling, the loop iterations are divided and assigned to PEs before execution; both division and assignment remain fixed during execution. This work considers static (block) scheduling, denoted STATIC, each PE being assigned a chunk size equal to the number of iterations N divided by the number of PEs P. STATIC incurs *minimum* scheduling overhead, compared to dynamic loop scheduling, and may lead to load imbalance for non-uniformly distributed tasks and/or on perturbed systems.

In *dynamic loop scheduling* (DLS), free and requesting PEs are assigned, via self-scheduling, loop iterations during execution. The DLS techniques can be categorized into *nonadaptive* and *adaptive* techniques. The nonadaptive DLS techniques considered in this work are: SS [19], FSC [14], GSS [20], FAC [12], and WF [11]. While STATIC represents one scheduling extreme, SS represents the other scheduling extreme. In SS, the size of each chunk is one loop iteration. This yields a high load balance with potentially very large scheduling overhead. FSC assigns loop iterations in chunks of fixed sizes, where the chunk size depends on the standard deviation of loop iteration execution times σ as an indication of its variation and the incurred scheduling overhead h. GSS assigns loop iterations in chunks of decreasing sizes, where the size of a chunk is equal to the number of remaining unscheduled loop iterations R divided by the number of PEs N. FAC employs a probabilistic modeling of loop characteristics (standard deviation of iterations execution time σ and their mean μ) to calculate batch sizes that maximize the probability of achieving a load balanced execution. A PE's chunk size is equal to the batch size divided by N. When this information (σ and μ) is unavailable, FAC is practically implemented to assign half of the remaining loop iterations R in a batch. WF divides a batch into unequally-sized chunks, proportional to the relative PE speeds (weights). The PEs weights need to be determined prior to the execution and do not change afterward. This work considers the *practical implementations* of FAC and WF. All nonadaptive DLS techniques account for variations in iteration execution times due to application characteristics.

The adaptive DLS techniques measure the performance of the application during execution and adapt the chunk calculation accordingly. Adaptive

DLS techniques include AWF [3], its variants [7]: AWF-B, AWF-C, AWF-D, AWF-E, and AF [2]. AWF is designed for time-stepping applications. It improves WF by changing the relative weights of PEs during execution by measuring their performance in each time step and updating their weights accordingly. AWF-B relieves the time stepping requirement in AWF, and measures the performance after each batch to update the PE weights. AWF-C is similar to AWF-B, where weight updates are performed upon the completion of each chunk, instead of a batch. AWF-D is similar to AWF-B, and considers the total chunk time (equal to the chunk iteration execution times plus the associated overhead of a PE to acquire the chunk) and all the bookkeeping operations to calculate and update the PE weights. AWF-B and AWF-C only consider the chunk iterations execution times. AWF-E is similar to AWF-C by updating the PE weights on every chunk. Yet AWF-E is also similar to AWF-D by also considering the total chunk time also. Unlike FAC, AF dynamically estimates the values of σ and μ during execution and updates them based on the measured performance of the PEs.

Loop scheduling in simulation. SimGrid [8] (SG) is a versatile event-based simulation toolkit designed for the study of the behavior of large-scale distributed systems. It provides ready to use application programming interfaces (API) to represent applications and computing systems through different interfaces: MSG (SG-MSG), SimDag (SG-SD), and SMPI (SG-SMPI). SG uses a simple, fast CPU computation model and verified network models [24] which render it well suited for the study of computationally-intensive distributed scientific applications.

Various studies have used SG to study the performance of applications with DLS techniques in different scenarios [4, 22, 23]. To attain high trustworthiness in the performance results obtained with SG, the implementation of the nonadaptive DLS techniques in SG-SD has been verified [18] by reproducing the results presented in the work that introduced factoring [12]. Also, the accuracy of simulative performance experiments against native experiments has recently been quantified [16]. This work employs the SG-SD interface to study the performance of scientific applications on a heterogeneous platform under perturbations.

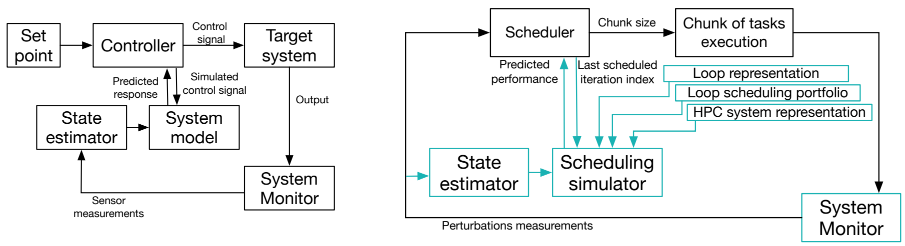
Related work. Robustness denotes the maintenance of certain desired system characteristics despite fluctuations in the behavior of its components or its environment [1], whereas, flexibility [22] denotes the robustness of DLS to variations in the delivered computational speeds. The performance of scientific applications under perturbations in the delivered computational speed is studied with nonadaptive DLS techniques [13, 25]. The robust scheduling of tasks with uncertain communication time was also considered using a multi-objective evolutionary algorithm [6] and to evaluate the flexibility of DLS [22]. The selection of the best performing DLS during execution was studied for OpenMP multi-threaded applications [26], and for time-stepping applications using reinforced learning [4]. Also, machine learning was used to create a portfolio of DLS robustness to variations in the delivered computational speed on a homogeneous system [23].

Scheduling solutions based on optimization techniques, e.g., genetic and evolutionary algorithms, can not adapt to perturbations during execution. None of the aforementioned efforts considered perturbations in network bandwidth and latency. This work complements the previous efforts by studying the performance of scientific applications using nonadaptive and adaptive DLS techniques under different perturbations (variations in delivered computational speed, network bandwidth, network latency) on a heterogeneous computing system. A new approach, namely *simulator in the loop* (SiL) is introduced, to dynamically select DLS techniques that improve the performance of applications on heterogeneous system under multiple sources of perturbations.

3 Simulator in the Loop (SiL)

The SiL is inspired by control theory, where a controller (scheduler) is used to achieve and maintain a desired state (load balance) of the system (parallel loop execution), as illustrated in Figure 1. The SiL concept is motivated by the well-known control strategy model predictive control (MPC) [21]. The MPC controller predicts the performance of the system with different control signals to optimize system performance. As shown in Figure 1(b), a call to the SiL simulator is inserted inside a typical scheduling loop. SiL leverages state-of-the-art simulation toolkits to estimate the performance of an application in a given execution scenario. The system monitor and estimator components read the system state during the execution and update the computing system representation accordingly. The above steps may be repeated several times during the execution of the loop, and this frequency can be aligned with the perturbations frequency or intensity.

The advantage of SiL is that it leverages the use of already developed



(a) A generic control system.

(b) Proposed SiL approach for loop scheduling.

Figure 1: The proposed *simulator in the loop* (SiL) approach for loop scheduling (b) is analogous to a typical control system (a). The components highlighted in mint color in (b) represent the SiL additions to a typical loop scheduling system.

state-of-the-art simulators to predict the performance dynamically during execution. The accuracy of the simulator and its prediction is strongly influenced by the representation of both applications and the systems in simulation as well as by the available subsystems models in the simulator [16]. For instance, the percent error between native and simulative executions for a given application (PSIA [10]) using the SG-SD interface was found to be between 0.95% and 2.99% [16]. It is expected that the accuracy and the speed of the simulators employed by SiL will improve as they are continuously being developed and refined. The cost of frequent calls to SiL can be amortized by launching parallel SiL instances to concurrently derive predictions for various DLS. Alternatively, this cost can be entirely mitigated by asynchronously calling SiL, concurrently to the application execution. Upon completion, SiL returns the recommended *best suited DLS technique* to the calling application, which can then directly use the recommended DLS to improve the application performance.

The system monitor and estimator components can be implemented with a number of system monitoring tools [9], such as `collectl`. Such tools can periodically be instantiated to measure PE and network loads and to update the system representation in the simulator. The measured chunk execution times can also be used to estimate the current PE computational speeds. The PE loads can be estimated and predicted using autoregressive integrated moving average [15].

4 Evaluation and Analysis

Experimental Design and Setup. The factorial design of experiments is presented in the following (cf. Table 1), together with the applications performance and a discussion thereof.

Table 1: Design of factorial experiments

Factors	Values	Properties
Applications	Problem size	N = 400,000 iterations
	PSIA	$[5.9 \cdot 10^7, 6.6 \cdot 10^7]$ FLOP per iteration
	Constant	$2.3 \cdot 10^8$ FLOP per iteration
	Uniform	$[10^3, 7 \cdot 10^8]$ FLOP per iteration
	Normal	$\mu = 9.5 \cdot 10^8$ FLOP, $\sigma = 7 \cdot 10^7$ FLOP, $[6 \cdot 10^8, 1.3 \cdot 10^9]$ FLOP per iteration
	Exponential	$\lambda = 1/3 \cdot 10^8$ FLOP, $[948, 4.5 \cdot 10^9]$ FLOP per iteration
	Gamma	$k = 2$, $\theta = 10^8$ FLOP, $[4.1 \cdot 10^6, 2.7 \cdot 10^9]$ FLOP per iteration
Loop scheduling	STATIC	Static
	SS, FSC, GSS, FAC, WF	Nonadaptive dynamic
	AWF-B, -C, -D, -E, AF	Adaptive dynamic
Computing system	miniHPC (heterogeneous HPC cluster)	22 Intel Broadwell nodes $(22 \cdot 20$ cores), relative core weight = 1.398
		4 Intel Xeon Phi KNL nodes $(4 \cdot 64$ cores), relative core weight = 0.316
		P = 224 heterogeneous (112 Broadwell + 112 KNL) cores
		P = 696 heterogeneous (440 Broadwell + 256 KNL) cores
Perturbations	Nominal conditions	no perturbations (np)
	PE availability	constant mild (pea-cm)
		constant severe (pea-cs)
		exponential mild (pea-em)
		exponential severe (pea-es)
	Bandwidth	constant mild (bw-cm)
		constant severe (bw-cs)
		exponential mild (bw-em)
		exponential severe (bw-es)
	Latency	constant mild (lat-cm)
		constant severe (lat-cs)
		exponential mild (lat-em)
		exponential severe (lat-es)
	All	constant mild (all-cm)
		constant severe (all-cs)
		exponential mild (all-em)
		exponential severe (all-es)
Experimentation	Native[a]	PSIA on 224 cores under no perturbations (online[b])
	Simulative	All applications on 224 cores under all perturbations (online[b])
		All applications on 696 cores under all perturbations

[a] Direct experiments on real HPC systems.
[b] Included in this arxiv.org submission, please download all data

Applications. This work considers a real-world application and five synthetic applications. The parallel spin-image algorithm [10] (PSIA), is an application from computer vision. The PSIA is algorithmically load imbalanced and the computational effort of a loop iteration depends on the input data. The performance of PSIA has been studied in prior work [10] and enhanced for a heterogeneous cluster by using nonadaptive DLS techniques. The total number of PSIA loop iterations is 400,000. To represent the PSIA in simulation, the number of floating point operations (FLOP) of each loop iteration is counted using PAPI [5] counters. In SG-SD, each loop iteration is represented as a task [16, 17]. Each of the five synthetic applications contains 400,000 parallel loop iterations, similar to the PSIA. The FLOP count in each loop iteration is assumed to follow five different probability distributions, namely: constant, uniform, normal, exponential, and gamma probability distributions. The probability distribution parameters used to generate these FLOP counts are given in Table 1.

Loop scheduling. Eleven loop scheduling techniques are used to assess the performance of the above six applications under test. These techniques represent a wide range of loop scheduling approaches, namely, *static* and *dynamic*. The dynamic loop scheduling (DLS) approach can further be distinguished into adaptive and nonadaptive. The DLS techniques can be implemented using centralized or decentralized execution and control approach. The decentralized control approach was found to scale better by eliminating a centralized master, and hence, the master-level contention [18]. The DLS implemented using the decentralized control approach is considered in this work.

Computing system. miniHPC[1] consists of 26 compute nodes: 22 nodes each with one dual socket Intel Xeon E5-2640 v4 (20 cores) configuration and 4 nodes each with one Intel Xeon Phi Knights Landing 7210 processor (64 cores). The total number of heterogeneous cores is 22 nodes $\times$ 20 cores per node + 4 nodes $\times$ 64 cores per node = 696 cores. All nodes are interconnected with Intel Omni-Path fabrics in a nonblocking two-level fat-tree topology.

Simulation. A computing system is represented in SG via an XML file denoted as `platform file`. SG registers each processor core from their representation as a `host` in the `platform file`. The computational speed of a processor core is estimated by measuring a loop execution time and dividing it by the total number of floating point operations included in the loop [16]. A Xeon core was found to be four times faster than a Xeon Phi core as indicated by the relative core weights (cf. Table 1). The network bandwidth and latency represented in the `platform file` are calibrated with the SG calibration procedure[2].

Perturbations. Three different categories of perturbations are considered

[1] miniHPC is a fully controlled non-production HPC cluster at the Department of Mathematics and Computer Science at the University of Basel, Switzerland.

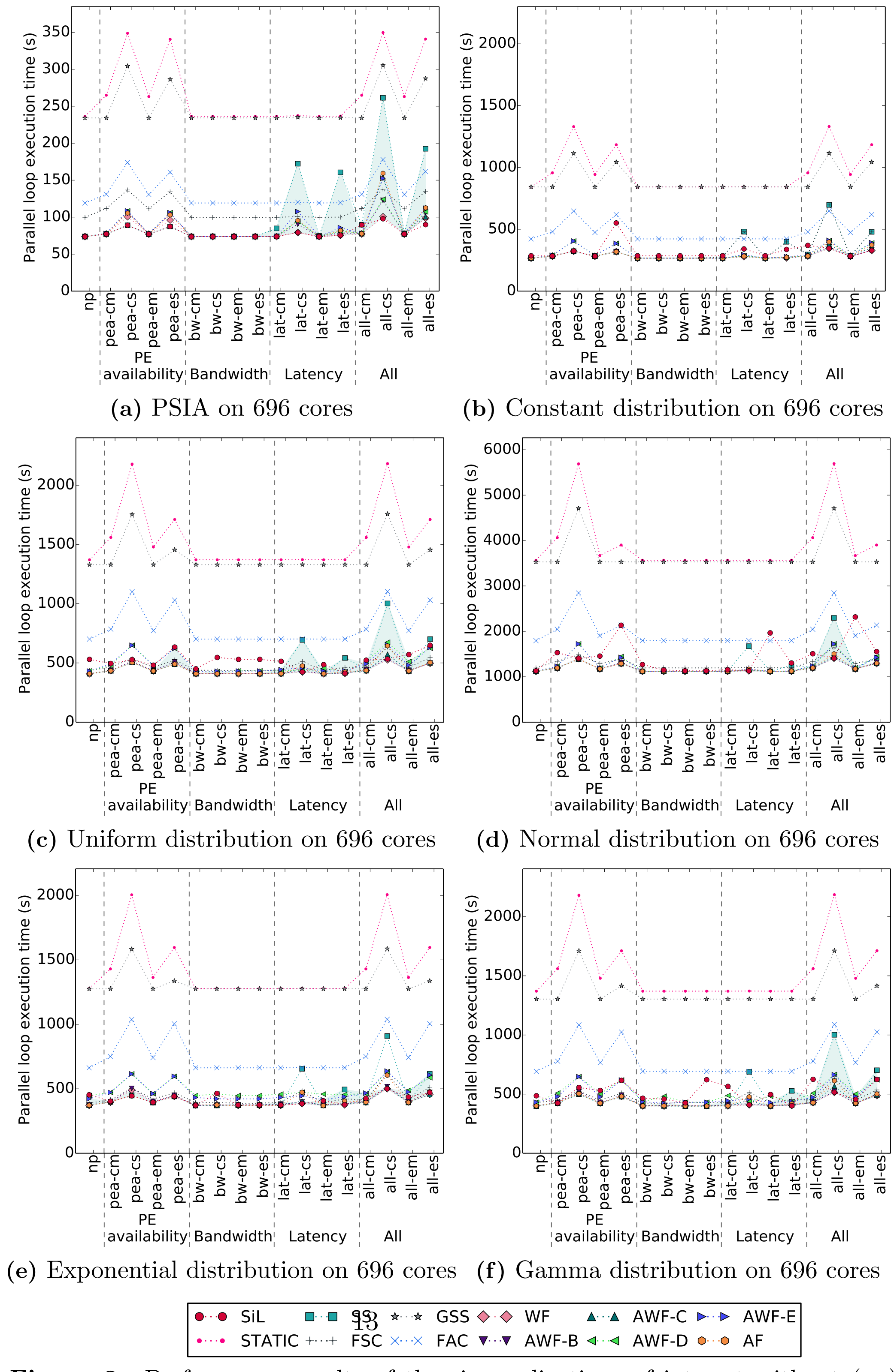
[2] http://simgrid.gforge.inria.fr/contrib/smpi-calibration-doc/

in this work, namely *delivered computational speed*, *available network bandwidth*, and *available network latency*. Two intensities are considered, mild and severe, for each category. Two scenarios are considered for each intensity, where the value of the delivered computational speed is either constant or exponentially distributed. All perturbations (cf. Table 1) are considered to occur periodically, with a period of 100 seconds where the perturbations affect the system only during 50% of the perturbation period. The network (bandwidth and latency) perturbations commence with the application execution, while the delivered computational speed perturbations begin 50 seconds after the start of the application. The PE availability to compute changes to 75% and 25% for the mild and severe intensities, respectively. The available network bandwidth and network latency change to 0.001% and 0.00001% for the mild and severe intensities, respectively. Another perturbation scenario is created by combining all perturbations from the other individual categories. All perturbations are enacted in SG during simulation via the `availability`, `bandwidth`, `latency`, and `platform` files.

Performance of Scientific Applications under Perturbations. The performance of the six applications of interest is shown in Figure 2. One can see that STATIC, FSC, GSS, and FAC perform poorly on heterogeneous systems. WF is well suited for scheduling on heterogeneous systems. However, it can not adapt to accommodate the variability in the system due to perturbations, especially perturbations in the delivered computational speed. SS is resilient to perturbations in the delivered computational speed of the PEs. However, it is significantly influenced by the network latency variations, as can be seen in Figure 2a "lat-cs" and "lat-es". Perturbations in the network bandwidth show a very small influence on performance, as the PEs only communicate loop iterations indices to calculate the start index of the next chunk. Therefore, the communicated messages are small.

The adaptive techniques perform comparably, with a slight advantage for AWF-C as can be seen in Figure 2e "all-cs" and in Figure 2a "pea-cs" and "all-es". However, in certain cases, other techniques outperform AWF-C. Specifically, WF outperforms AWF-C in Figure 2a "lat-cs" and "all-cs". *These results suggest that no single DLS outperforms all other techniques in all execution scenarios.* Therefore, the best strategy is to dynamically select a DLS based on the current application and system states.

In this work, SiL is called every 50 seconds to select the best performing DLS. A closer analysis of the SiL-based results reveals that it resulted in the



(a) PSIA on 696 cores

(b) Constant distribution on 696 cores

(c) Uniform distribution on 696 cores

(d) Normal distribution on 696 cores

(e) Exponential distribution on 696 cores

(f) Gamma distribution on 696 cores

Figure 2: Performance results of the six applications of interest without (np) and with (the rest) perturbations using SiL and eleven loop scheduling techniques on 696 heterogeneous cores. The mint color shaded regions denote the upper and lower bounds of the performance with SiL if only one DLS technique were selected during execution in the particular execution scenario.

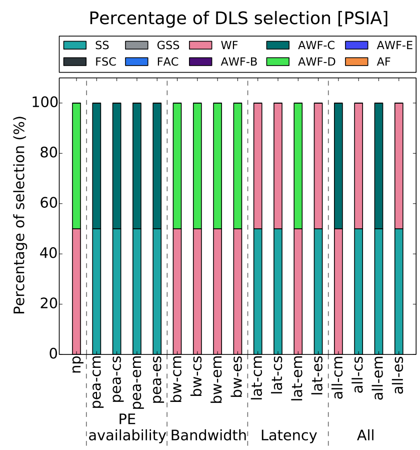


Figure 3: DLS selection results for the PSIA application. DLS techniques, such as FSC, GSS, and FAC are not selected due to their poor predicted performance with SiL.

smallest execution time in most execution scenarios, especially for PSIA, as shown in Figure 2a. The PSIA execution with SiL in the "all-es" scenario outperformed all other techniques, as the best DLS technique was changed during the execution according to the execution scenario. In other cases, the application performance with SiL was slightly slower than the minimum execution time achieved by other DLS. This is due to the fact that loop scheduling is, by definition, non-preemptive and the execution of already scheduled loop iterations can not be preempted to be resumed with the newly selected DLS.

Discussion. The advantage of the SiL approach is to dynamically select the DLS that is predicted to achieve the best performance. A combination of two or more DLS techniques throughout the application execution may result in a shorter execution time than that achievable by any single DLS technique alone as can be seen in Figure 2a "all-es". The SiL selected WF for the first 50 seconds in "all-es", as can be seen in Figure 3. After 50 seconds, the network was no longer perturbed, and SiL selects the SS technique to balance the load and achieve a better performance than any single DLS technique. The simulative performance results of the PSIA on 224 heterogeneous cores (112 Broadwell cores and 112 KNL cores) have been verified by native experimentation under the *no perturbation* execution scenario. The raw results and details of the DLS selection for all the applications can be found online[3]. The native experimentation of application performance in other execution scenarios is planned as immediate future work. In certain cases, such as "all-em" in the application with normally distributed tasks, the SiL-based execution did not yield the best performance, due to the fact that DLS is non-preemptive. The DLS techniques selected via SiL can be used as guidelines for a given application, computing system, and perturbation scenario. The SiL approach can proactively select the best suited DLS before any perturbations act on the system, when perturbations can be predicted in advance. The study and prediction of perturbations on HPC systems need further examination, as perturbations in HPC shared resources are inevitable. The cost of the SiL simulation depends on the problem size and the system size. Specifically, simulating the execution of 20,000 iterations on 9 PEs with SG-SD executing on an Intel Broadwell E5 processor, with CentOS 7.2 operating system, required 0.34 seconds on average, whereas, it required 3.48 seconds for simulating the execution of 200,000 iterations on the same number of PEs. These costs can be amortized or entirely mitigated by calling the simulator asynchronously to the parallel loop execution.

5 Conclusion and Future Work

A new control-theoretic inspired approach, namely simulator in the loop (SiL), was introduced to dynamically select a DLS that achieves the best performance, in an effort to answer the question of which DLS technique will achieve improved performance under unpredictable perturbations. The performance of six applications is studied under perturbations and insights on the resilience of the DLS techniques to perturbations are provided. The performance results confirm the hypothesis that no single DLS technique can achieve the best performance in all the considered execution scenarios. Using the SiL approach improved the performance of applications in most considered experiments. SiL leverages state-of-the-art simulators to select the DLS predicted to result in the best performance of an application under perturbations. The SiL can be asynchronously launched concurrently to the application execution. The results show that in the case of a system perturbed via multiple sources, a combination of two or more DLS techniques may result in improved performance than that achievable by any single DLS

[3]Included in this arxiv.org submission, please download all data

alone, such as the performance of the PSIA in "all-es" execution scenario. However, due to applications being non-preemptively scheduled, changing the used DLS during the execution may not result in the best performance. Further work is planned to realize and evaluate the performance of the SiL approach using native experimentation. Furthermore, experiments to investigate and enhance the performance of SiL, in terms of improving the DLS selection strategy and the period between SiL calls also planned as future work.

References

[1] Ali, S., Maciejewski, A.A., Siegel, H.J., Kim, J.K.: Measuring the Robustness of a Resource Allocation. IEEE Transactions on Parallel and Distributed Systems **15**(7), 630–641 (2004)

[2] Banicescu, I., Liu, Z.: Adaptive Factoring: A Dynamic Scheduling Method Tuned to the Rate of Weight Changes. In: Proceedings of the High Performance Computing Symposium. pp. 122–129 (2000)

[3] Banicescu, I., Velusamy, V., Devaprasad, J.: On the Scalability of Dynamic Scheduling Scientific Applications With Adaptive Weighted Factoring. Cluster Computing **6**(3), 215–226 (2003)

[4] Boulmier, A., Banicescu, I., Ciorba, F.M., Abdennadher, N.: An Autonomic Approach for the Selection of Robust Dynamic Loop Scheduling Techniques. In: Proceedings of 16th International Symposium on Parallel and Distributed Computing. pp. 9–17 (2017)

[5] Browne, S., Dongarra, J., Garner, N., Ho, G., Mucci, P.: A Portable Programming Interface for Performance Evaluation on Modern Processors. International Journal of High Performance Computing Applications **14**(3), 189–204 (2000)

[6] Canon, L.C., Jeannot, E.: Evaluation and Optimization of the Robustness of DAG Schedules in Heterogeneous Environments. IEEE Transactions on Parallel and Distributed Systems **21**(4), 532–546 (2010)

[7] Cariño, R.L., Banicescu, I.: Dynamic Load Balancing With Adaptive Factoring Methods in Scientific Applications. Journal of Supercomputing **44**(1), 41–63 (2008)

[8] Casanova, H., Giersch, A., Legrand, A., Quinson, M., Suter, F.: Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms. Journal of Parallel and Distributed Computing **74**(10), 2899–2917 (2014)

[9] Ciorba, F.M.: The Importance and Need for System Monitoring and Analysis in HPC Operations and Research. In: Proceedings of the 3rd bwHPC-Symposium: Heidelberg 2016. pp. 7–16. heiBOOKS (2017)

[10] Eleliemy, A., Mohammed, A., Ciorba, F.M.: Efficient Generation of Parallel Spin-images Using Dynamic Loop Scheduling. In: Proceedings of the 19th IEEE International Conference for High Performance Computing and Communications Workshops. pp. 34–41 (2017)

[11] Flynn Hummel, S., Schmidt, J., Uma, R., Wein, J.: Load-sharing in Heterogeneous Systems via Weighted Factoring. In: Proceedings of the Annual ACM Symposium on Parallel Algorithms and Architectures. pp. 318–328. ACM (1996)

[12] Flynn Hummel, S., Schonberg, E., Flynn, L.E.: Factoring: A method for scheduling parallel loops. Communications of the ACM **35**(8), 90–101 (1992)

[13] García-González, L.A., García-Jacas, C.R., Acevedo-Martínez, L., Trujillo-Rasúa, R.A., Roose, D.: Self-Scheduling for a Heterogeneous Distributed Platform. In: Proceedings of the International Conference on Parallel Computing. pp. 232–241 (2017)

[14] Kruskal, C.P., Weiss, A.: Allocating Independent Subtasks on Parallel Processors. IEEE Transactions on Software Engineering **SE-11**(10), 1001–1016 (1985)

[15] Mehrotra, R., Banicescu, I., Srivastava, S., Abdelwahed, S.: A Power-aware Autonomic Approach for Performance Management of Scientific Applications in a Data Center Environment. In: Handbook on Data Centers, pp. 163–189. Springer (2015)

[16] Mohammed, A., Eleliemy, A., Ciorba, F.M., Kasielke, F., Banicescu, I.: Experimental Verification and Analysis of Dynamic Loop Scheduling in Scientific Applications. In: Proceedings of the 17th International Symposium on Parallel and Distributed Computing. p. 8 (2018)

[17] Mohammed, A., Eleliemy, A., Ciorba, F.M.: A Methodology for Bridging the Native and Simulated Execution of Parallel Applications. Poster at ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (2017)

[18] Mohammed, A., Eleliemy, A., Ciorba, F.M.: Performance Reproduction and Prediction of Selected Dynamic Loop Scheduling Experiments. In: Proceedings of the 2018 International Conference on High Performance Computing and Simulation. p. 8 (2018)

[19] Peiyi, T., Pen-Chung, Y.: Processor Self-Scheduling for Multiple-Nested Parallel Loops. In: Proceedings of the International Conference on Parallel Processing. pp. 528–535 (1986)

[20] Polychronopoulos, C.D., Kuck, D.J.: Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers. IEEE Transactions on Computers **100**(12), 1425–1439 (1987)

[21] Rawlings, J.B.: Tutorial: Overview of Model Predictive Control. IEEE Control Systems **20**(3), 38–52 (2000)

[22] Sukhija, N., Banicescu, I., Srivastava, S., Ciorba, F.M.: Evaluating the Flexibility of Dynamic Loop Scheduling on Heterogeneous Systems in the Presence of Fluctuating Load Using SimGrid. In: Proceedings of the 27th IEEE International Parallel and Distributed Processing Symposium Workshops. pp. 1429–1438 (2013)

[23] Sukhija, N., Malone, B., Srivastava, S., Banicescu, I., Ciorba, F.M.: Portfolio-based Selection of Robust Dynamic Loop Scheduling Algorithms Using Machine Learning. In: Proceedings of the 28th IEEE International Parallel and Distributed Processing Symposium Workshops. pp. 1638–1647 (2014)

[24] Velho, P., Legrand, A.: Accuracy Study and Improvement of Network Simulation in the SimGrid Framework. In: Proceedings of the 2nd International Conference on Simulation Tools and Techniques. p. 10 (2009)

[25] Yang, Y., Casanova, H.: Rumr: Robust Scheduling for Divisible Workloads. In: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing. pp. 114–123 (2003)

[26] Zhang, Y., Voss, M., Rogers, E.: Runtime Empirical Selection of Loop Schedulers on Hyperthreaded SMPs. In: Proceedings of the 19th International Parallel and Distributed Processing Symposium. p. 10 (2005)