Optimizing resource utilization on large-scale systems through predictive communication modeling

Nikela Papadopoulou
National Technical University of Athens
Athens, Greece, 15780

Georgios Goumas
National Technical University of Athens
Athens, Greece, 15780

Nectarios Koziris
National Technical University of Athens
Athens, Greece, 15780

ABSTRACT

On the path to exascale, supercomputers will grow to host hundreds of million of cores and various complex heterogeneous processing elements, yet even today, users fail to leverage the existing compute power of large-scale systems, as large classes of typical HPC applications are bound by non-scalable communication phases. The ability to predict the execution time of parallel applications can assist users, compilers, runtime systems and schedulers with decision-making for optimal resource utilization, performance optimizations, power saving and resilience. In this work, we utilize predictive models for communication time of HPC applications to enable users in their decisions on resource allocation. We construct predictive models for two supercomputers, Vilje and Piz Daint, and we evaluate the predictive ability of our models on a mini-application, a 3D-Jacobian relaxation. We use this application and our predictions for its communication time to detect optimal execution configurations for varying numbers of cores, up to 8192. Our evaluation shows significant accuracy in the prediction of optimal execution configurations, in terms of cores, nodes and processes per node. Using the same predictions, we also detect configurations that allow users to save up on their corehour budget.

1. INTRODUCTION

The growth of communication overheads is a limiting factor to attaining petaflop performance for several applications that run on large-scale systems [5]. Applications like PDE solvers on structured and unstructured grids, multi-scale and multi-model codes and unstructured graph processing, which are typically strong-scaled, will retain their growing communication overheads on exascale systems, unless there is a radical shift in the programming paradigm. Therefore, to avoid the dissipation of valuable computing resources and power on application that do not scale up, there is an imperative need for ahead-of-execution prediction of the expected application performance and scalability, as well as for effective decision-making on resource allocation and utilization.

Typically, HPC applications share their execution time between computation and communication phases. Computation scalability is easy to predict, through inspection of their performance on similar platforms or simulation, as it takes place on a contained environment, namely several identical, isolated multi-core platforms and/or accelerators. On the other hand, predicting communication scalability, especially when it comes to projecting on more cores, poses many challenges, which stem from the distributed, massive and shared nature of the interconnection network on large-scale systems. In practice, the users have barely any means to predict the scalability of their applications. It is common among supercomputer users to overestimate the potential ability of their application to scale and request excessive numbers of cores from the system to run their applications. As a result, users undergo long waiting hours and consume their corehour budget on non-scalable configurations. At the same time, they dissipate valuable compute resources.

Analytical models for communication time, such as [1], make a simplified assumption of a linear dependence of communication performance on the roundtrip time and thus fail to capture complex communication effects that arise on large scale. An empirical approach by Bhatele et al. in [3] achieves projection of communication performance on large scale on BlueGene/Q with the assistance of network performance counters. Shudler et al. in [7] build linear models to project the scalability of applications to exascale.

In this paper, we show how empirical performance models for communication time can enable a user to make optimal decisions on the execution configuration of her application, taking into account communication scalability. We use our methodology, described in [6], to construct predictive models for communication time on two supercomputers, Vilje and Piz Daint. Using these predictive models, we explore the ability to predict optimal node and core configurations, to find the number of cores and the configuration that result in the lowest execution time, before scalability breaks, and to select configurations that allow the user to save up on her corehour budget, along with system resources. We use a mini-application, a 3D-Jacobian relaxation, with a typical 3D-stencil communication pattern, to evaluate the predictive ability of our models in the aforementioned decision-making scenarios. Our experimental results show remarkable accuracy in predicting critical configurations, even when projecting to 4 times on Vilje and 8 times on Piz Daint more cores than those with which the models have been trained.

2. BACKGROUND

2.1 Communication performance issues

Interconnection networks for large-scale systems come with various architectures, topologies, configurations and system software, constituting a complex entity. Large-scale applications are mapped onto these complex systems and produce a communication configuration which can vary dramatically in between different executions. In the following paragraphs,

we discuss the effects of the system and the application on communication performance.

Network topologies are a critical design parameter of interconnection networks. The topology per se defines the *bisection bandwidth* and the *path diversity* between a source and a destination, determines the number of nodes attached to switches and classifies networks as direct and indirect. Besides the topology, network technology defines several architecture - related parameters, such as the *latency* and the *link bandwidth*. Other critical parameters for communication performance source from the network interface architecture.

Parallel applications request a set of resources like number of cores/nodes, memory size, etc. Schedulers allocate a system partition based on the current resource allocation and certain optimization targets (either system- or application-centric), producing a specific *process mapping* of the application on the system. Depending on the topology and scheduler decisions, the allocation shape can vary drastically. Alternate process mappings affect the *dilation*, the link *congestion* [4], the *mixture of intranode and internode communication*, the length of network queues and buffers on nodes and switches [3]. Subsequent application executions do not maintain the process mapping, a fact that often leads to significantly diverse mappings and communication times. Implementation details of the communication library (typically MPI) also impact communication performance. Protocol and transport selection, as well as algorithmic or system-level optimizations, may differentiate communication behavior between MPI implementations. OS noise and interference from nearby jobs have also been reported to degrade the communication performance of parallel applications [2].

At the application level, there exists a number of features that constitute its communication profile and greatly affect its communication time. Application processes may communicate irregularly or in synchronized communication phases. In this work we focus on applications that communicate in distinct phases as is a typical case for large classes of scientific applications. The *total data volume* exchanged within each phase and the *communication pattern* (point-to-point vs collective) are critical factors. The *number* and *size of messages* per process and whether these features remain constant across all processes are also of high importance.

Every application execution on a target system creates a unique *communication configuration* involving architectural, system-level and application-level parameters as discussed in the previous paragraphs. As multiple streams of data of an application flow through the interconnection network during its *communication phase*, network resources are shared among the participating processes and all kinds of *contention* effects may occur. In addition to link sharing, the switches that participate in communication are also stressed. High dilation may also induce contention, or occasionally add up positively to the path diversity. In summary, an accurate performance model for the communication of parallel applications requires an aggregation of all the aforementioned metrics, features and effects.

2.2 Building machine-learning models for communication time

In our methodology for the construction of predictive models for communication time, we attempt to abstract the complex effects of communication on large-scale systems and combine them in simple and accurate models, built with machine-learning techniques. In particular, we define quantifiable, descriptive metrics for communication, which we draw from the application communication profile and the process mapping. We use metrics that denote the amount of data and messages that are communicated between the processes of an application and metrics that describe the shape of the allocation upon which the processes are mapped. To abstract the complex effects of communication at scale, we utilize a single benchmark to correlate these metrics with communication time. Based on the benchmark results, we identify relationships between the metrics and communication time, i.e. the target variable of our predictive models, and construct multiple variable regression models for communication time, which are trained with the data collected through benchmarking.

3. COMMUNICATION PERFORMANCE MODELS FOR LARGE-SCALE SYSTEMS

In this section, we describe the two systems for which we construct communication performance models. We provide a brief description of our metrics for communication performance modeling, present our models for predicting communication time on *Vilje* and follow the methodology described in [6] to construct models for communication time on *Piz Daint*. We evaluate the predictive ability of the models on a 3D-Jacobian relaxation application, with a 3D-halo communication pattern that often appears in HPC applications.

3.1 Target Systems

The *Vilje* supercomputer at NTNU is an SGI system of 1404 Intel Xeon-E5 dual eight-core nodes. It is interconnected with InfiniBand FDR in an enhanced hypercube topology, with redundant links at the lower dimensions of the hypercube. The nodes of *Vilje* are organized as following: the basic component of the topology is the IRU. Each IRU hosts 18 nodes and two InfiniBand switches and forms the first dimension of the hypercube. Four IRUs form a single rack. The total number of racks of *Vilje* is 19.5. The default MPI library for *Vilje* is a component of the SGI Message Passing Toolkit. The *Piz Daint* supercomputer at CSCS is a Cray XC30 system, consisting of 5752 nodes equipped with Intel Xeon-E5 eight-core nodes and NVIDIA GPUs. The interconnect is the proprietary Cray Aries. The nodes of *Piz Daint* are organized in a dragonfly topology, as following: the topology forms 14 groups, which are all-to-all connected; each group contains six chassis, each containing sixteen Cray Aries SoCs. Four nodes are connected on a Cray Aries SoC. All nodes within a group are all-to-all connected. The default MPI setup for *Piz Daint* is Cray MPICH.

3.2 Descriptive metrics for communication

To model communication time, we utilize metrics related to the application communication profile and its mapping on the underlying system. First, we define metrics that are purely derived from the application communication profile. These are the average message size **S**, the average number of messages **M** and the process traffic **PT**, i.e. the traffic (in bytes) that is created by each process. We also define two metrics for the allocation size, the number of nodes **N** and the number of processes per node **PPN**. Once the processes of an application are mapped on the given allocation, there is a certain amount of data in bytes and messages that

Table 1: Communication Performance Models on Vilje: Additive terms and their coefficients

Area Ia $S \leq 4kB$ and $NT \leq 128kB$		Area Ib $S > 4kB$ and $NT \leq 128kB$		Area II $NT > 128kB$	
Terms	Coefficients	Terms	Coefficients	Terms	Coefficients
Intercept	6.6111×10^{-6}	Intercept	0	Intercept	0
$PPN \times NT$	5.8226×10^{-12}	S	7.7787×10^{-10}	S	-3.5128×10^{-11}
$S \times NT$	-1.42×10^{-13}	SW/R	1.2193×10^{-06}	$PPN \times S$	-1.3962×10^{-11}
$R \times NT$	-5.967×10^{-11}	$S \times SW/R$	-6.1059×10^{-11}	$SW \times S$	1.8077×10^{-11}
$PPN \times NI$	2.1315×10^{-08}	NT	2.8278×10^{-10}	$PPN \times S \times SW$	-1.0744×10^{-12}
$S \times NI$	1.0384×10^{-09}	V	1.2398×10^{-12}	NT	2.5808×10^{-10}
$R \times NI$	8.1344×10^{-08}	$S \times NT$	-5.7408×10^{-15}	$PPN \times NT$	-3.8401×10^{-12}
$PPN \times S \times NT$	-5.9903×10^{-16}	$SW/R \times NT$	2.6589×10^{-11}	$S \times NT$	4.5466×10^{-18}
$PPN \times R \times NT$	1.2994×10^{-12}	$S \times V$	3.6211×10^{-18}	$SW \times NT$	1.3029×10^{-11}
$S \times R \times NT$	1.2228×10^{-14}	$SW/R \times V$	-1.8116×10^{-13}	$PPN \times S \times NT$	1.5495×10^{-19}
$PPN \times R \times NI$	-4.7418×10^{-09}	$S \times SW/R \times NT$	6.9513×10^{-16}	$PPN \times SW \times NT$	3.5872×10^{-13}
$PPN \times S \times R \times NT$	-3.7129×10^{-16}	$S \times SW/R \times V$	2.3784×10^{-18}	$S \times SW \times NT$	-1.6864×10^{-19}
				$PPN \times S \times SW \times NT$	2.1983×10^{-20}

is injected from the node to the network. We define and use two metrics, the node traffic **NT**, i.e. the data injected from a node to the network in bytes and node injection **NI**, i.e. the number of messages injected from a node to the network. Note that, the two metrics exclude the amount of data that is exchanged intra-node, i.e. between processes that reside on the same node. We also define the total communication volume **V** and the total injected messages **VI**, which are the amount of data in bytes and messages respectively exchanged over the network during an application communication phase. All the aforementioned metrics are architecture-agnostic and utilized without modifications across any system for the purpose of modeling communication time. In addition, we define and utilize a few metrics that describe the allocation shape, which are specific to each architecture. We describe these metrics and how they specify for each system in the following sections.

3.3 Modeling communication on Vilje

To model communication on Vilje, we define three additional metrics for the allocation shape, based on the characteristics of the architecture and topology of the system. The metrics are the number of switches **SW** and racks **R** in an allocation for an application execution and their ratio, i.e. switches per rack **SW/R**. We use these metrics as indicators of possible hot-spots, the dilation of the allocation and the allocation density.

Following the methodology we describe in [6], we collected a training set for the modeling process by executing a benchmark on Vilje for various allocations, ranging from 8 up to 128 nodes, 1 up to 16 processes per node, 1 up to 4 messages per processes and messages of length from 64B up to 16MB. All the configurations were executed twice, to collect diverse values for the number of switches and racks in the allocation and their ratio. Based on this training set, we examined correlations and scatterplots between the defined metrics and communication time and decided to parameter space in three areas. First, we split according to the node traffic into Area I (**NT** $\leq$ 128kB) and Area II (**NT** $>$ 128kB). Then we split Area I according to the message size into Area Ia (**S** $\leq$ 4kB) and Area Ib (**S** $>$ 4kB). For each of the three areas, we selected the form of the model, making use of our observations on the relationships between our metrics and communication time and we built a robust multiple variable regression model, using the class *LinearModel*, available in Matlab R2013a. The model terms and their coefficients for each area are presented in Table 1.

Table 3: Communication patterns and configurations used for evaluation of prediction accuracy

	Vilje	Piz Daint
N	16-512	16-1024
PPN	1-16	1-8
Domain Size	128^3, 256^3, 512^3, 1024^3, 2048^3	
Iterations	256	
$\#executions$	3	

3.4 Modeling communication on Piz Daint

Extending our work in [6], we apply the same methodology to model communication time on Piz Daint. As on Vilje, to capture the properties of the dragonfly topology of Piz Daint, we define metrics for the allocation shape. Thes metrics are the number of Aries SoCs **A**, chassis **C** and groups **G** in the allocation, as well as their ratios, namely the number of nodes per Aries SoCs **A/C**, Aries SoCs per chassis **A/C** and chassis per group **C/G**. Similarly to Vilje, we use these metrics as indicators of hot-spots, dilation, path diversity and allocation density.

To collect the training set, we executed our benchmark on Piz Daint for various allocations of 8 up to 128 nodes, 1 up to 8 processes per node, 1 up to 8 messages per process and message sizes ranging from 64B up to 16MB. The benchmark was executed twice, on different allocations each time, to collect diverse values for the number of Aries SoCs, chassis and groups and their ratios. After inspecting scatterplots of the various metrics against communication time, we divided our training set in three areas, where training set points exhibit common relationships to our metrics. First, we split our training set according to the message size into Area I (**S** $\leq$ 8kB) and Area II (**S** $>$ 8kB). Then, we split Area I into Areas Ia (**PT** $\leq$ 8kB) and Ib (**PT** $>$ 8kB). For each of the three areas, we built a robust multiple variable regression model on Matlab R2013a. The model terms and their coefficients for each area are presented in Table 2.

3.5 Evaluating prediction accuracy

To evaluate the prediction accuracy of our models for communication time, we use a 3D-Jacobian relaxation (hereafter denoted as *Jacobi3D*) application, which exposes a 3D-halo communication pattern, where 3D-subdomains exchange 2D-faces with their six neighboring processes. We

Table 2: Communication Performance Models on Piz Daint: Additive terms and their coefficients

Area Ia $S \leq 8kB$ and $PT \leq 8kB$		Area Ib $S \leq 8kB$ and $PT > 8kB$		Area II $S > 8kB$	
Terms	Coefficients	Terms	Coefficients	Terms	Coefficients
Intercept	1.816×10^{-5}	Intercept	2.0604×10^{-5}	Intercept	7.0526×10^{-5}
NI	5.0144×10^{-8}	PPN	-1.7625×10^{-5}	PPN	-2.7899×10^{-5}
N/A	-2.927×10^{-6}	NT	1.9318×10^{-10}	NT	2.2196×10^{-10}
$NI \times N/A$	3.2788×10^{-8}	NI	2.7981×10^{-7}	$NT \times PPN$	1.4061×10^{-11}
C	-6.4923×10^{-7}	A/C	7.5479×10^{-7}		
$NI \times C$	1.3872×10^{-8}	$PPN \times NT$	5.3778×10^{-13}		
$N/A \times C$	2.9738×10^{-7}	$PPN \times NI$	-1.9800×10^{-8}		
$NI \times C \times N/A$	-1.8538×10^{-9}	$NT \times A/C$	2.7822×10^{-12}		

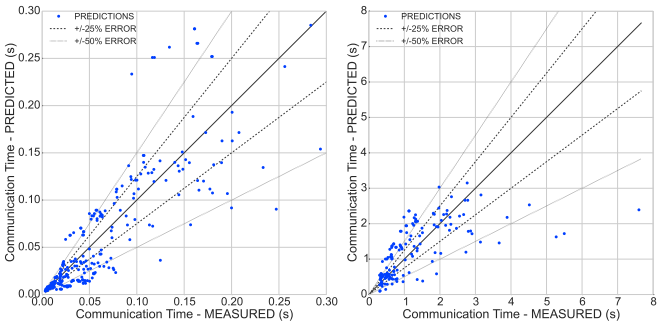


Figure 1: Communication time predictions on Vilje

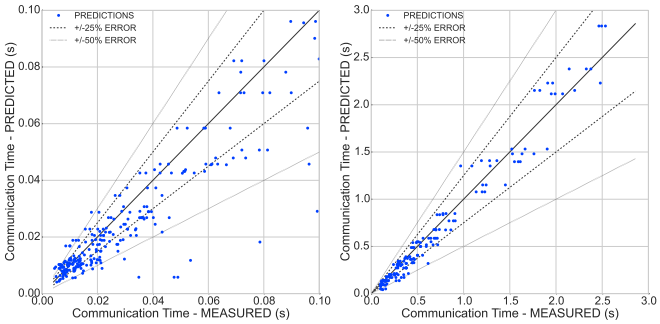


Figure 2: Communication time predictions on Piz Daint

executed the application on the two systems, Vilje and Piz Daint, for various problem sizes, on multiple nodes and cores configurations. Additionally, we collected measurements from multiple executions which give diverse allocation shapes. We provide a detailed description of the collected measurements in Table 3. We predict the communication time of the application, using the models presented in the previous section for Vilje and Piz Daint, for all the aforementioned configurations. We note that, while our models have been trained with configurations up to 2048 cores on Vilje and 1024 cores on Piz Daint, our evaluation set includes configurations on 8192 cores on Vilje and Piz Daint, namely 4 times and 8 times higher numbers of cores than those included in the training set. In this way, we also evaluate the ability of our models to project communication scalability.

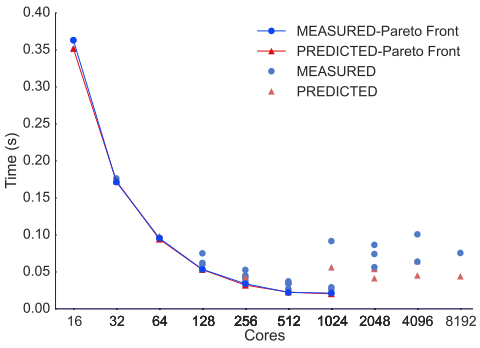
Figures 1 and 2 show the scatterplots for the predicted communication time of *Jacobi3D*, compared to the measured communication time, for all five problem sizes and all nodes/processes per node configurations, as well as for all three executions on different allocations, on Vilje and Piz Daint respectively. On Vilje, 41.3% of our predictions lie within a $\pm 25\%$ error margin and 68.2% lie within a $\pm 50\%$ error margin. We note, however, that, on Vilje, our models do not capture some communication effects, as we observe a few extreme underpredictions and overpredictions. Underpredictions, in their majority, correspond to configurations of high node (>128) and core counts (>1024) and small problem sizes, i.e. where many messages of extremely small sizes are exchanged over the network. Such configurations are not part of the training set. Overpredictions also correspond to configurations with high core counts (>1024), beyond our training set, on large problem sizes, where possibly the effect of the message size is overestimated. Both behaviors can be interpreted as an inability of the models to project on higher core counts than those included in the training set. On the other hand, our models for Piz Daint exhibit remarkable accuracy consistently in all configurations and project well to higher core counts than those included in the training set; 68.8% of predictions lie within a $\pm 25\%$ error margin and 92.6% lie exhibit errors within $\pm 50\%$. Some observed underpredictions cannot be grouped with some characteristic and thus we argue that they are outlier cases, where communication time is higher than expected due to system noise or interference from other applications.

4. COMMUNICATION-AWARE DECISION-MAKING

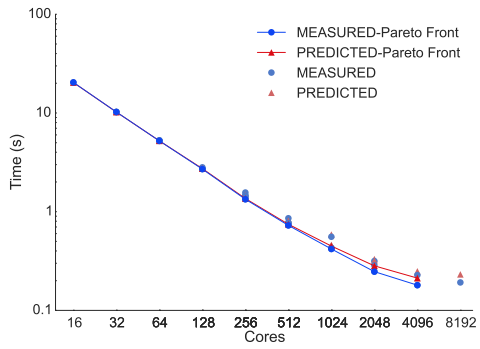
The ability to predict the execution time of parallel applications is valuable for decision-making regarding resource allocation and utilization. More importantly, the utility of a prediction model is evaluated by its ability to predict critical points in various decision-making scenarios. We examine a few such scenarios, where our prediction models could prove useful, and test the prediction accuracy.

One problem for supercomputer users is to find the configuration of nodes and processes per node that minimizes execution time. A user can acquire a number of cores in multiple possible pairs of nodes and processes per node, but has barely any means to predict which configuration results to the lower execution time. This prediction would guide the users to the configurations that minimize execution time and help them avoid configurations that do not scale well. It would also potentially save resources, as the users could release cores of a node or use them for a different purpose, e.g. to improve computation performance only, with a hybrid MPI/OpenMP implementation.

To address this problem, we formulate it as a problem of

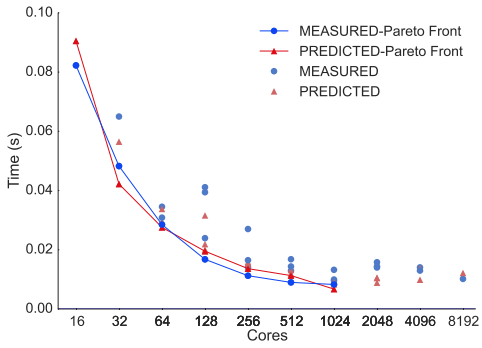


(a) Pareto front for *Jacobi3D*-128^3

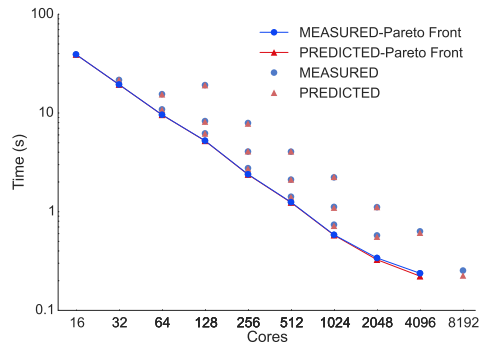


(b) Pareto front for *Jacobi3D*-512^3

Figure 3: Optimal configurations for *Jacobi3D* on Vilje



(a) Pareto front for *Jacobi3D*-128^3



(b) Pareto front for *Jacobi3D*-1024^3

Figure 4: Optimal configurations for *Jacobi3D* on Piz Daint

minimizing simultaneously the number of cores, in different configurations, and the execution time. This set is known as the Pareto front and marks an optimal set of choices, in the sense that the user is neither better nor worse off by making either of these choices. We assume that there exists a perfect predictor for the computation time of *Jacobi3D* (in practice, we use the observed computation time) and utilize our prediction models to estimate communication time on the two systems, Vilje and Piz Daint. We compute the Pareto fronts for the execution time and cores using our predictions and compare against the Pareto fronts for the observed execution time, for each problem size of *Jacobi3D*.

Figures 3 and 4 show two examples of the Pareto fronts on Vilje and Piz Daint respectively, for various problem sizes. In all cases, the predicted Pareto front is close (or identical) to the real Pareto front, i.e. our models are capable of detecting those configurations that minimize execution time. In addition, in the cases we present, scalability breaks at a certain number of cores, as indicated by the points constituting the real Pareto front. The predicted Pareto fronts detect the scalability breaks with remarkable accuracy. Tables 4 and 5 summarize the results of the Pareto front points for measured and predicted values of execution time and cores. We denote as "Matches" the common Pareto front points for the two Pareto fronts, which also correspond to identical configurations. On both systems, the Pareto fronts for measured and predicted points match perfectly in 2 out of the 5 cases, while at most 3 points are mispredicted for the remaining 3 cases.

Table 4: Pareto Fronts for the Cores/Execution Time Minimization on Vilje

Jacobi3D	Pareto Points		
Problem Size	Measured	Predicted	Matches
128^3	7	7	5
256^3	7	7	6
512^3	9	9	9
1024^3	10	10	7
2048^3	10	10	10

Table 5: Pareto Fronts for the Cores/Execution Time Minimization on Piz Daint

Jacobi3D	Pareto Points		
Problem Size	Measured	Predicted	Matches
128^3	7	7	4
256^3	10	10	8
512^3	10	10	9
1024^3	9	9	9
2048^3	9	9	9

The Pareto fronts for cores and execution time offer additional interesting information, which can be deployed for the optimization of resource utilization. First, as we mentioned before, they identify scalability breaks. If, for a given number of cores, no configuration is included in the Pareto front, then the application no longer scales after this number of cores. Also, the point with the highest core count included in the Pareto front corresponds to the configuration that leads to the minimum execution time. The opti-

Table 6: Minimum Time Configuration Prediction on Vilje

Problem Size	Measured(NxPPN)	Predicted(NxPPN)	Result
128^3	512×2	512×2	True
256^3	512×2	512×2	True
512^3	512×8	512×8	True
1024^3	512×16	512×16	True
2048^3	512×16	512×16	True

Table 7: Minimum Time Configuration Prediction on Piz Daint

Problem Size	Measured(NxPPN)	Predicted(NxPPN)	Result
128^3	128×8	128×8	True
256^3	1024×8	1024×8	True
512^3	1024×8	1024×8	True
1024^3	1024×4	1024×4	True
2048^3	1024×4	1024×4	True

Table 8: Corehours minimization on Vilje

	Minimum corehours configurations			Corehours	
Problem Size	Measured(NxPPN)	Predicted(NxPPN)	Result	Measured	Predicted
128^3	32×1	32×1	True	0.001524	0.001535
256^3	16×1	16×1	True	0.0115	0.0113
512^3	16×1	16×1	True	0.0907	0.0905
1024^3	32×1	32×1	True	0.7168	0.7161
2048^3	32×1	32×1	True	5.724	5.723

Table 9: Corehours minimization on Piz Daint

	Minimum corehours configurations			Corehours	
Problem Size	Measured(NxPPN)	Predicted(NxPPN)	Result	Measured	Predicted
128^3	16×1	32×1	False	0.000364	0.000375
256^3	32×1	32×1	True	0.00243	0.00240
512^3	256×1	256×1	True	0.02035	0.02061
1024^3	1024×1	1024×1	True	0.1659	0.1647
2048^3	512×1	512×1	True	1.3716	1.3634

mality condition is the same for both problems. Finally, all points at the Pareto front minimize the core-hour consumption by the user for the application for a given number of cores. Therefore, one point at the Pareto front denotes the minimal consumption of corehours.

Tables 6 and 7 show the predicted and actual minimum time configurations for each tested problem size of *Jacobi3D* on Vilje and Piz Daint. For all cases, the minimum time configuration is accurately predicted on both systems. We should note that scalability breaks are also accurately detected when they occur, that is in 3 cases on Vilje (for problem sizes 128^3, 256^3 and 512^3) and in 3 cases on Piz Daint (for problem sizes 128^3, 1024^3 and 2048^3). Tables 8 and 9 show the predicted and actual minimum corehours and the configurations where they occur. There is a single mismatch on Piz Daint for problem size 128^3, where however the error in corehours is 3%.

5. CONCLUSIONS AND FUTURE WORK

In this work, we explored the potential of predictive performance models for communication time of HPC applications in decision-making scenarios, regarding optimal resource allocation and utilization by the user. Following a machine-learning approach for communication performance modeling, we constructed predictive models for communication time on two large supercomputers, Vilje and Piz Daint. We evaluated the overall predictive ability of the models on a representative HPC mini-application, a 3D-Jacobian relaxation. We used our predictions for communication, while assuming a perfect predictor for computation, to detect optimal execution configurations for this application, using Pareto fronts for the execution time and cores. Our experimental results show great accuracy in detecting optimal configurations, perfect accuracy in predicting the number of cores and the configuration that results in the lowest execution time and possible scalability breaks. Using the same Pareto fronts, we also detected the configurations that result in the lowest possible corehour consumption, that would allow the users to save up on their corehour budgets.

In future work, we intend to explore advanced machine-learning techniques for the automation of the modeling process and we will work towards improving the overall accuracy of our predictive models. We also aim at modeling collective and/or irregular communication. On optimizing resource utilization, we target to explore decision-making scenarios that arise at the schedulers/resource managers of large-scale computing systems.

6. ACKNOWLEDGEMENT

This research was supported in part with computational resources at the Norwegian University of Science and Technology provided by NOTUR (http://www.notur.no) and in part by a grant from the Swiss National Supercomputing Center under project ID g83. Nikela Papadopoulou has received funding from IKY fellowships of excellence for postgraduate studies in Greece-SIEMENS program.

7. REFERENCES

[1] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP: incorporating long messages into the LogP model—one step closer towards a realistic model for parallel computation. In *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, pages 95–105. ACM, 1995.

[2] A. Bhatele, K. Mohror, S. H. Langer, and K. E. Isaacs. There goes the neighborhood: performance degradation due to nearby jobs. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, page 41. ACM, 2013.

[3] A. Bhatele, A. R. Titus, J. J. Thiagarajan, N. Jain, T. Gamblin, P.-T. Bremer, M. Schulz, and L. V. Kale. Identifying the culprits behind network congestion.

[4] T. Hoefler and M. Snir. Generic topology mapping strategies for large-scale parallel architectures. In *Proceedings of the international conference on Supercomputing*, pages 75–84. ACM, 2011.

[5] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, et al. Exascale computing study: Technology challenges in achieving exascale systems. 2008.

[6] N. Papadopoulou, G. Goumas, and N. Koziris. A machine-learning approach for communication prediction of large-scale applications. In *2015 IEEE International Conference on Cluster Computing*, pages 120–123. IEEE, 2015.

[7] S. Shudler, A. Calotoiu, T. Hoefler, A. Strube, and F. Wolf. Exascaling your library: Will your implementation meet your expectations? In *Proceedings of the 29th ACM on International Conference on Supercomputing*, pages 165–175. ACM, 2015.