Cluster Comput (2017) 20:2725-2747

Predictive Communication Modeling for HPC Applications

Nikela Papadopoulou · Georgios Goumas · Nectarios Koziris

Received: 11 October 2016 / Revised: 15 February 2017 / Accepted: 14 March 2017

Abstract In this paper, we present a methodology for predictive modeling of communication of HPC applications. Communication time depends on a complex set of parameters, relevant to the application, the system architecture, the runtime configuration and runtime conditions. To handle this complexity, we define features that can be extracted from the application, the process mapping and the allocation shape ahead of execution, deploy a single benchmark to sweep over the parameter space and develop predictive models for communication time on two supercomputers, Vilje and Piz Daint, using different subsets of our features, machine-learning methods and training sets. We compare the predictive power of our models on two common communication patterns and one application, for various problem sizes, executions and runtime configurations, ranging from a few dozen to a few thousand cores. Our methodology is successful across all tested communication patterns on both systems and exhibits high prediction accuracy and goodness-of-fit, scoring 23.98% in *MMRE*, 0.942 in *RCC* and 61.43% in $Pred_{0.25}$ on Vilje and 21.31%, 0.940 and 66.57% respectively on Piz Daint, with models that are applicable just-in-time ahead of the execution of an HPC application.

Keywords Predictive Modeling · Communication Time · MPI Applications · Supercomputers · Machine Learning

1 Introduction

As supercomputers grow in cores and computational capacity, the challenge of delivering high levels of performance at extreme scales is becoming constantly harder. Exascale systems are expected to have thousands, or even millions of cores in various configurations; however large classes of parallel applications that require tight collaboration between their processing elements, will fail to exploit a decent fraction of this enormous compute power. Parallel performance depends on the scalability of computation and communication, with computation scaling remarkably well as it utilizes multiple nodes, cores, vector units, heterogeneous accelerators or specialized hardware. Contrarily, as the number of the participating processing elements increases, the cost of communication may rise enough to outweigh computation scalability, thus rendering communication a serious obstacle towards exascale performance.

Predictions of communication time can support code optimizations and effective resource allocation on large-scale systems. The ability to decide between different configurations (e.g. number of nodes, processes per node, networking components in use) has been shown to critically affect performance [27], energy [6] and resilience [54]. Moreover, a communication prediction model can be used to support code optimization and tuning for a wide range of communication optimization techniques like computation / communication overlapping [26], message compression [21], communication avoiding [16], hybrid MPI/OpenMP [19, 47]. Finally, communication performance models are a core component in trace-driven simulators for parallel applications, such as LogGOPSim [31] and SimGrid [11]

Communication performance is dependent on a multitude of factors, and as such, its accurate modeling poses significant challenges. The communication time for a specific communication pattern on a specific system heavily depends on the traffic pattern, produced by the mapping of the application communication graph on the system allocation, dedicated to the execution of the application, the system architecture and its network topology properties. The traffic created by a single application can be enough to induce contention on network components and congestion on network

School of Electrical and Computer Engineering
National Technical University of Athens
E-mail: {nikela, goumas, nkoziris}@cslab.ece.ntua.gr

links, with equivalent delays, however on large-scale systems, where multiple applications co-execute, they compete for shared resources, interfere and are able to cause unexpected delays to the communication time of each other. The setup of each system, its communication software, operating system and resource management components influence communication time of applications in a complex manner. Also, computation phases of applications affect communication time, as they can overlap with communication or skew communication time due to load imbalance.

Existing approaches for communication modeling include analytical, empirical and semi-empirical models. State-of-art analytical models [29, 15, 2] focus on a local view of communication and rely on round-trip time measurements to assess communication parameters, such as sender and receiver overheads, latency and transfer time. Their purpose is to facilitate hardware / software co-design and communication software design and optimizations. They can provide predictions for communication time ahead of execution, however, as they ignore the network topology, allocation and any resource-sharing effects, they are not able to accurately capture the communication time of application communication phases. A pure empirical approach to communication performance modeling is presented by Jain et al. [37] and Bhatele et al. [9]. They utilize network performance counters and machine-learning techniques to build communication models for the BlueGene/Q. This approach attains significantly higher accuracy than analytical models on load-balanced applications. On the downside, as their models rely on post-execution performance counter measurements, they can only be used for explanatory and not predictive purposes. Ghavari et al. [24, 25] focus on predictive modeling and provide a semi-empirical solution for communication time prediction, extending Hockney's latency-bandwidth analytical model with empirical parameters which capture topology attributes, contention and congestion effects on each system. We evaluate this semi-empirical approach in our experimental results. In our previous work [45], we follow an empirical approach and construct multiple-variable regression models for communication time prediction on a large-scale system, based on descriptive metrics from the application and its mapping on the system. We extend this approach with a multitude of additional features for communication time, automated machine-learning methods for model building and test it on an additional architecture.

In this paper we propose a methodology for the construction of empirical predictive models for point-to-point communication of HPC applications. Our approach aims at predicting the time for an entire communication phase of an application and offers predictions before the execution of the application. To our knowledge, this is the first approach to provide communication modeling that predicts the communication of a wide set of point-to-point communication patterns without any customization, ahead of execution, delivering significant prediction accuracy on two different architectures. To accomplish this, we consider features that affect communication performance taken from the application characteristics, its mapping on the target system and the configuration of the underlying system architecture (Section 3.2). We construct a simple benchmark to create a generic training set for use across all tested point-to-point communication patterns (Section 3.3) and deploy machine-learning methods to construct a number of predictive models for communication time, that expose different levels of accuracy and applicability (Sections 3.4, 3.5). We apply our methodology on two large supercomputers with state-of-art interconnection networks, i.e. Vilje, an InfiniBand SGI Altix system and Piz Daint, a Cray XC30 system (Section 4).

We evaluate our predictive models for various communication phases of representative HPC kernels and real-world applications, for multiple configurations and problem sizes and achieve remarkably accurate predictions across this wide experimental setup. Throughout this evaluation, we also validate two significant assumptions that have driven our work: a) generic benchmarking that makes our methodology applicable to multiple applications is adequate for training an accurate model on both platforms, b) a predictive communication model needs to consider multiple features that well describe the traffic pattern, in order to incorporate the complex effects of communication over a large-scale system. We compare the predictive ability of our models against a baseline analytical model and semi-empirical models (Section 4.2) following the methodology proposed by Ghavari et al. [24, 25]. Our empirical approach outperforms any other approach in terms of prediction accuracy, achieving *MMRE* (Mean Magnitude of Relative Error), *RCC* (Rank Correlation Coefficient) and $Pred_{0.25}$ (percentage of predictions with relative error within the range of $\pm 25\%$) scores of 23.98%, 0.942 and 61.43% on Vilje and 21.31%, 0.940 and 66.57% on Piz Daint.

2 Problem definition

Our work targets the prediction of communication time for communication phases of applications, as seen by all involved processes. In the following paragraphs, we present our target platforms, the challenges of predicting communication time on large-scale systems and the parallel applications that we target.

2.1 Target platforms

Our modeling methodology targets large-scale systems with high-end interconnection networks. In this work we consider two platforms, *Piz Daint*, a Cray XC30 system, built with the proprietary Cray Aries interconnect that implements a dragonfly topology, and *Vilje*, an SGI Altix system, built with InfiniBand FDR that implements an enhanced hypercube topology. The two platforms are quite diverse in their network architecture and topology and are typical representatives of about 52% of current top-tier supercomputers[1]. InfiniBand is the most popular interconnection technology, used by 44.4% of the systems in the Top500 list, while Cray Aries is steadily gaining ground on the Top500 list, due to its promising topology.

Piz Daint[2], the Cray XC30 installation at the Swiss National Supercomputing Center (CSCS), comprises 5272 nodes, each equipped with an 8-core Intel Sandy Bridge CPU, an NVIDIA Tesla K20X GPU and 32GB of memory. The core component of the fabric is the *Aries SoC*, with four NICs connecting four nodes, forming a blade. Sixteen blades are accommodated on a *chassis*. A pair of cabinets, each hosting three chassis, forms a *group* of the network. Aries chips within a group are interconnected in an all-to-all topology via the chassis backplane and copper cables, while the 14 groups of Piz Daint are interconnected in an all-to-all topology through optical cables of slightly lower bandwidth. Congestion control is achieved through adaptive packet-level routing: a non-minimal path is selected if the estimated link load is lower than that of the minimal path. More details on Cray Aries can be found in [3]. The default MPI setup for Piz Daint is Cray MPICH.

Vilje[3] is an SGI Altix ICE X system at the Norwegian University of Science and Technology (NTNU), comprising 1404 nodes of two 8-core Intel Sandy Bridge CPUs and 32GB of memory. The basic building block of the system is the Individual Rack Unit (IRU), which hosts 18 nodes and two 36-port FDR InfiniBand *switches*. Each IRU is a node of the hypercube topology. IRUs are stacked in groups of eight and interconnected in a 3D-hypercube topology to form a *rack*. The 19.5 racks of the system form a dual-rail, 8D enhanced hypercube, where redundant links are used to connect switches at the lower dimensions of the hypercube, providing higher overall bandwidth. Congestion control is achieved through a signaling mechanism: the source node is notified to throttle its injection of packets whenever congestion is detected on a switch on the path to the destination. The default MPI setup for Vilje is SGI MPI implementation.

2.2 Challenges of communication prediction

An effective predictive model maximizes *accuracy* and *applicability*. Accuracy captures the ability of the model to approximate the actual communication time with tolerable errors. Applicability refers to the scope of use of the model, i.e. the capacity of the model to support a specific task. In other words, applicability is about the availability in time of the prediction and the relevant ease-of-use. The tuning factor for the two attributes is the *awareness* of the model to the factors that affect communication performance. High awareness of these factors is expected to improve accuracy while curtailing applicability, as more information is required, and information becomes progressively available in the lifetime of an application. The optimal trade-off between accuracy and applicability is defined by the purpose of the model.

Communication performance is a result of the coaction between the application, the interconnection network, the system software and the mapping of the application on the system. Capturing the complexity of this coaction in a model is highly challenging and one needs to carefully glean appropriate features, with varying awareness, from a huge space of parameters. The application exposes its own *communication pattern* and a specific *problem size* and *decomposition*. The interconnection network comes with a specific backbone *architecture* consisting of network interface cards, switches, routers, links, etc, with their number determined by the system scale and *topology*. The *system software* includes high and low level communication protocols, the OS and network drivers, protocol-switching thresholds, lengths of buffers and queues for communication and more. For one execution of an application, the *scheduler* of the system undertakes the *allocation* of resources and the *placement* of the processes. This application *mapping* interacts with the system to produce a specific *traffic pattern* which ultimately determines communication performance.

A minimal communication model would require application - centric and architecture - centric information (e.g. communication pattern and topology respectively) and would be applicable at *static time* (or early at runtime, once the input parameters of the application are set). However, to achieve the highest possible accuracy, a model should incorporate additional critical information being part of the traffic pattern, that more accurately captures the distribution of the data flow within the network. The shaping of the traffic pattern demands knowledge of the *process mapping* and the *allocation shape*, which becomes available *just-in-time* before the execution of the application. Alternate process mappings may produce very diverse traffic patterns, with a varying outcome of communication performance, as the distribution of traffic changes and different points of the network are stressed [37].

Although knowing the traffic pattern is necessary in order to incorporate the complex effects and interactions of the application and the system into a prediction model, even with this knowledge at hand, accurate communication time prediction is undermined by non-deterministic sources of time variability that occur on real-world systems, which can

[1] http://www.top500.org (June 2016)
[2] http://www.cscs.ch/computers/piz_daint_piz_dora/index.html
[3] https://www.hpc.ntnu.no/display/hpc/Vilje

only be monitored at runtime. Allocations that populate more networking components are more prone to interference with co-running applications [8, 38] and may suffer from performance degradation, due to contention on these components. On Vilje, performance degradation may occur due to switch sharing. On Piz Daint, time variability may occur due to non-deterministic routing. In addition to inter-application contention, system noise, originating from the nodes due to OS services or due to a low byte-to-flop ratio [20], causes a delay that may propagate throughout the communication phase or be overlapped by regular communication synchronization delays [30]. Time variability that occurs due to these runtime effects cannot be captured by any predictive model and thus, prediction accuracy will be subject to this limitation.

The challenges in modeling and predicting communication time increase by the presence of computation and the interleaving of computation and communication phases in real-life applications. First, load imbalance in computation causes direct and indirect delays in communication [10]. While direct delays can possibly be estimated if computation time is known, a direct delay in communication of one process due to load imbalance propagates throughout the communication phase, skewing the communication phase. The total delay can only be identified and characterized with post-mortem trace analysis, as in [51, 10]. Second, computation and communication may be overlapped. In this case, the time consumed on a communication phase depends on the available time for hiding communication, on the amount of computation that is overlapped and on data dependencies between computation and communication phases [48]. Modeling communication time when computation and communication are overlapped also requires knowledge of exact times of request arrivals for send / receive operations [13]. To our knowledge, there is no prior work in communication time prediction that considers applications with load imbalance or computation / communication overlapping, as in such cases, the notion of communication as a phase collapses. The effect of computation on communication time can be taken into account only in holistic approaches for scalability modeling of parallel applications, as in [50]. Some recent studies perform trace analysis to identify application phases, including communication phases, either by identifying logical phases which do not appear as temporal phases due to some source of delay [36], or by segmenting traces into meaningful clusters of events [1].

HPC applications expose numerous communication patterns, with different characteristics, which then result into more or less intricate traffic patterns. Based on the type of communication operations, communication patterns may be *point-to-point* or *collective*. For collective operations, the pattern (number of peer processes, number of messages per process) is predefined, however the message size can vary

```
compute()
for ms in MessagesToSend do
    pack(ms)
end for
for mr in MessagesToReceive do
    MPI_Irecv(mr)
end for
for ms in MessagesToSend do
    MPI_Isend(ms)
end for
MPI_Waitall(MessagesToSend, MessagesToRecv)
for mr in MessagesToReceive do
    unpack(mr)
end for
```

Fig. 1 Pseudocode for each MPI process in the application model

for vector collective operations and they may be overlapped with computation in their non-blocking version, in which case collective communication does not occur as a phase. Point-to-point patterns that appear in HPC applications can be roughly classified into three large categories. The first category consists of patterns where all processes exchange roughly the same number of messages with a specific set of peer processes and communication occurs as a phase. These patterns appear in applications solving problems on structured or unstructured grids and are known as *nearest-neighbor* patterns, e.g. stencils, halo exchanges or particle exchanges. The maximum number of messages exchanged results from the numerical method, the geometry of the grid and the problem decomposition, while the message sizes can vary, also due to the geometry or the type of the problem. They are found in numerous applications, such as conjugate gradient solvers, adaptive mesh refinement, the multigrid cycle, molecular dynamics, hydrodynamics and more. The second category regards patterns that arise in *pipelined wavefront* applications, as is the LU decomposition and particle transport codes. These patterns are usually regular in the number of communicating processes and maximum number of messages, as are nearest-neighbor patterns, however communication is overlapped with computation, due to the structure of the problem, and thus, communication does not occur as a phase. The third category consists of patterns that arise in applications which involve *sparse matrices or graphs*, as is the Sparse Matrix-Vector multiplication, conjugate gradient solvers and graph traversals. These patterns are the most irregular in terms of communicating processes, number of messages per process and message sizes, as those depend on the structure of the sparse matrix or the graph, and communication may or may not occur as a phase.

2.3 Target applications

To cope with the extremely high challenges of communication time prediction, we focus our work on the family of HPC applications, where communication occurs as a phase.

We consider applications with point-to-point communication patterns, where all MPI processes exchange roughly the same number of messages, as are *nearest-neighbor* communication patterns. Such patterns arise in a multitude of HPC applications that solve problems on structured and unstructured grids, such as QCD simulations, weather simulations, hydrodynamics, molecular dynamics and others. In this way, we are able to provide a methodology that addresses the core challenges of communication modeling, and provide accurate models for a wide class of HPC application phases. As our target is to predict communication time for a communication phase, we consider applications where computation effects do not influence communication time. In Section 6, we discuss how our methodology can be extended to predict irregular and collective communication patterns.

A kernel for a simplified application model with a single computation and communication phase is listed as pseudocode in Fig. 1. As most HPC applications are iterative, multiple distinct communication phases may appear within a single iteration and then repeatedly appear in every iteration. The end-goal of our work is to predict communication time for any communication phase i of an application $\hat{t}_i$ and provide a prediction for the total communication time of the execution $\hat{T} = \sum_i \hat{t}_i$. We need to note that, although our work is motivated by and applied to typical HPC applications running on large-scale supercomputers, our approach is also applicable to any type of application with point-to-point communication, as is, for example, the case of large-scale graph processing with the bulk-synchronous model [52] in frameworks like Hama[4] and Pregel [40].

3 Predictive communication modeling

As discussed in Section 2, the parameter space for communication performance is large and complex. The first step towards communication time modeling is to enumerate and quantify communication-descriptive features. We investigate features that can lead to predictive models, applicable just-in-time before the application execution, thus features which can be extracted up to just after the allocation of processes (and before the actual execution of the application). Then, we employ a benchmarking step to collect a set of measurements which will serve as a training set for the model construction. In the third step, we decide upon effective machine-learning methods, perform feature selection and build models with different numbers of features, methods and training sets. The specifics of this methodology are presented in the following paragraphs.

[4] http://hama.apache.org

3.1 Machine-learning preliminaries

3.1.1 Supervised learning

We assume there exists a vector of features $X = (x_1, x_2, ..., x_k)$ and a function f which determine the communication time t of an application, namely $t = f(X) + \varepsilon$. The end-goal of our work is to synthesize an approximation $\hat{f}$ of the function f to predict communication time $\hat{t} = \hat{f}(X)$ for any known or unknown input X, with an error $\varepsilon = y - \hat{y}$ that we can tolerate. The task falls under the category of *supervised learning*, where a learning algorithm utilizes a training set $Tr = \{(X_i, t_i), i = 1, ..., n\}$ to learn *by example* the approximation $\hat{f}$ [23]. Since the output t is continuous, the task also falls under the category of *regression*.

3.1.2 Feature selection

For each system we examine, we consider a plethora of features taken from the application and its mapping on the system. However, all these features are not necessarily significant for the prediction of communication time. When single-variable modeling is pursued, one needs to identify only one important feature, highly-correlated with the output. In the case of multiple variable modeling, systematic feature selection can assist in understanding the data and improving prediction accuracy [12]. Techniques for feature selection include association filtering and recursive feature elimination, and opt to eliminate redundant or irrelevant features which would add redundant information or noise to the model. The result of feature selection is a vector of features $X' = (x_1, x_2, ..., x_{k'}), k' < k$.

3.1.3 Methods for regression

There exist multiple methods for regression, with the simplest building *linear models* of the form $\hat{t}(X'') = b_0 + \sum_{i=1}^{k''} b_i x_i''$, where the elements of X'' include features from the original vector X, interactions between them, polynomial or non-linear transformations and possibly dummy variables. This method requires the specification of the function form by the user, while the coefficients are then computed through least squares. A linear model can be very effective for prediction, however it may be arbitrarily complex in terms and branches, especially in the case of multiple variables. Moreover, the user is responsible for identifying all underlying relationships between the features at hand and the output (i.e. in our case, communication time). On the other hand, linear models provide a closed form function, where the relationship between a feature and communication time is clear, while ensemble methods only provide ranking scores for the selected features.

Ensemble methods based on decision trees are more recent methods which have been proven to work well for regression problems. Decision trees are utilized by ensemble methods [55] as weak learners. Bagging methods, such as random forests and extremely randomized trees, build multiple decision trees simultaneously which are then averaged to reduce variance. Boosting methods, such as AdaBoost and Gradient Boosting Machines subsequently build weak estimators, with each new one focusing on points neglected by the previous one, in order to reduce bias. The main advantage of ensemble methods is that they provide a black-box approach to regression.

3.1.4 Evaluation metrics

To evaluate a model for communication time, we focus on its ability to predict communication time for any given communication pattern and execution configuration with a minimum error. For this purpose, we consider and examine several evaluation metrics. In our workflow, a model predicts the communication times $\hat{t}_i, i = 1, ..., m$ for a set of points $T = \{(X_i, t_i), i = 1, ..., m\}$, collected from benchmark or application runs on the target system. The relative prediction error is then defined as:

$$e_i = (\hat{t}_i - t_i)/t_i$$

We examine the distribution of the relative prediction errors, along with the minimum, median and maximum relative error values. We also take into account the Mean Magnitude of Relative Errors *MMRE* [14], as a numeric alternative to the distribution of relative errors:

$$MMRE = \frac{\sum_{i=1}^{m} |e_i|}{m}$$

To assess the accuracy and goodness-of-fit of a predictive model, we examine the percentage of predictions at level 0.25, $Pred_{0.25}$:

$$Pred_{0.25}(\%) = \frac{\#predictions\ with\ |e_i| \leq 0.25}{\#predictions} * 100\%$$

and the rank correlation coefficient *RCC*:

$$RCC = \sum_{1 \leq i \leq m} \sum_{1 \leq j \leq m} concordant_{ij} / \frac{m(m-1)}{2}$$

where

$$concordant_{ij} = \begin{cases} 1, if\ \ t_i < t_j\ \ and\ \ \hat{t}_i < \hat{t}_j \\ 1, if\ \ t_i > t_j\ \ and\ \ \hat{t}_i > \hat{t}_j \\ 0, otherwise \end{cases}$$

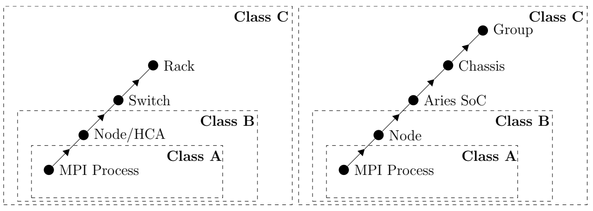


Fig. 2 Classes of features for the two systems, (a) Vilje and (b) Piz Daint. Class A features capture traffic from the processes, class B features capture traffic from the nodes, while class C features differentiate for the two systems and capture traffic on the upper levels of the topology. Notice there are more levels on Piz Daint than on Vilje.

Table 1 Class A Features: Cross-Platform

Feature	Name	Description
n	Nodes	Number of nodes of the allocation
ppn	Processes Per Node	MPI processes per node of the allocation
l	Message Length	The length (in bytes) of messages sent by each process [max]
PD	Process Data	Data (bytes) sent by each process [max]
PM	Process Messages	Number of messages sent by each process [max]

$Pred_{0.25}$ ranges from 0 to 100% and *RCC* ranges from 0 to 1, with higher values denoting higher accuracy and goodness-of-fit. $Pred_{0.25}$ measures prediction accuracy at a fixed threshold for the relative error (in our case, 0.25), while *RCC* measures how well the ordering of the data is predicted. A high value of $Pred_{0.25}$ indicates that the accuracy of the model is high for the partical set of points: most of the relative errors concentrate around 0%. A high value of *RCC* implies that the model is able to distinguish between different communication configurations; if one communication configuration leads to higher communication time than another, a model with a high *RCC* score also predicts higher communication time for the first communication configuration.

3.2 Classes of features

In our first step towards predictive communication modeling, we define quantifiable features for the application communication profile, traffic pattern and allocation shape for a given execution setup on the underlying system. We divide our features into three classes by their level of awareness to the application communication profile, its mapping and the system architecture. This division allows us to construct predictors with different degrees of applicability and potentially prediction accuracy. *Class A* features are collected from the application communication profile and its execution configuration. Values for these features can be extracted at static time with minimal effort and thus any predictor built upon class A features has high applicability, e.g. it can support decision making regarding application design and algorithmic optimizations. *Class B* augments class A with features related to the mapping of the processes of the application on the given node allocation for the specific execution configuration. Class B features measure the "local" traffic pattern, at the node level, but are unaware of the node allocation and the system architecture. Finally, *Class C* features sketch the traffic pattern from the application side, as well as the allocation shape. These features can be extracted only at runtime, after the system scheduler provides the node allocation, just-in-time before the execution of the application. Thus, any model built with Class C features is still predictive, incorporates nonetheless high awareness to various parameters that affect communication performance.

The majority of the defined features capture the traffic (in bytes) and message rate (number of messages) injected from processes to nodes and to upper levels of the interconnection network. Fig. 2 demonstrates an abstraction of how the classes are specified for the two machines, exposing different levels of awareness of the system's topology and organization. All features related to data and messages capture the outgoing traffic in bytes and number of messages from the processes of the application on system components (nodes, switches, Aries SoCs etc.) utilized by the allocation for the specific execution. Class A features are presented in Table 1. These features restrict themselves to knowledge extracted from the application, namely the application communication profile (message length, process data and messages) and the size of the allocation, the number of nodes and processes per node, which are provided by the user. Class B features, presented in Table 2 are aware of the mapping of the application on the allocated cores and nodes, thus include the intranode data and messages that may stress the memory system, internode data and messages that may induce contention due to limited injection bandwidth or limited capacity of network interface cards [9], as well as the total data and messages, denoting the total internode communication volume of the application. Features of classes A and B are cross-platform. Class C features for Vilje are presented in Table 3. Apart from data and messages for switches and racks of the allocation, which can reveal congestion on intermediate switches, we also consider the number of switches, racks, the average number of nodes per switch and the average number of switches per rack, as descriptive features of the allocation shape, which is usually irregular on Vilje. Similarly, we define Class C features for Piz Daint in Table 4. The switches and racks of Vilje are substituted by the Aries SoCs, the chassis and groups of Piz Daint. As in Piz Daint routing is adaptive and packets select a minimal or non-minimal path, depending on current link congestion, we additionally define the group-to-group data and messages, as indicative of excessive inter-group traffic that could stress the optical link between two groups and induce non-minimal routing. We should note that values for features like *ppn*, *n*, *sw* or *g*, that refer to the allocation are constant for an application execution, while the values for all other features are constant for a communication phase.

Table 2 Class B Features: Cross-Platform

Feature	Name	Description
ND	Node Data	Data (in bytes) injected from a node to the network [min, avg, max]
NM	Node Messages	Messages injected from a node to the network [min, avg, max]
iND	Intranode Data	Data (in bytes) sent intranode [min, avg, max]
iNM	Intranode Messages	Messages sent intranode [min, avg, max]
TD	Total Data	Data (in bytes) injected by all nodes of the allocation
TM	Total Messages	Messages injected by all nodes of the allocation

Table 3 Class C Features: Vilje

Feature	Name	Description
sw	Switches	Number of switches where the nodes of the allocation reside
r	Racks	Number of racks where the nodes of the allocation reside
n/sw	Nodes per Switch	Nodes per switch in the allocation [avg]
sw/r	Switches per Rack	Switches per rack in the allocation [avg])
SD	Switch Data	Data (in bytes) injected from a switch to the network [min, avg, max[
SM	Switch Messages	Messages injected from a switch to the network [min, avg, max]
iSD	Intra-Switch Data	Data (in bytes) sent between nodes attached on the same switch [min, avg, max]
iSM	Intra-Switch Messages	Messages sent between nodes attached on the same switch [min, avg, max]
RD	Rack Data	Data (in bytes) sent from a rack [min, avg, max]
RM	Rack Messages	Messages sent from rack [min, avg, max]
iRD	Intra-Rack Data	Data (in bytes) sent between switches residing on the same rack [min, avg, max]
iRM	Intra-Rack Messages	Messages sent between switches residing on the same rack [min, avg, max]

Table 4 Class C Features: Piz Daint

Feature	Name	Description
c	Chassis	Number of chassis where the nodes of the allocation reside
g	Groups	Number of groups where the nodes of the allocation reside
c/g	Chassis per Group	Chassis per group in the allocation [avg]
AD	Aries Data	Data (in bytes) injected from an Aries SoC to the network [min, avg, max]
AM	Aries Messages	Messages injected from an Aries SoC to the network [min, avg, max]
iAD	Intra-Aries Data	Data (in bytes) sent between nodes attached on the same Aries SoC [min, avg, max]
iAM	Intra-Aries Messages	Messages sent between nodes attached on the same Aries SoC [min, avg, max]
CD	Chassis Data	Data (in bytes) sent from a chassis [min, avg, max]
CM	Chassis Messages	Messages sent from a chassis [min, avg, max]
iCD	Intra-Chassis Data	Data (in bytes) sent between Aries SoCs residing on the same chassis [min, avg, max]
iCM	Intra-Chassis Messages	Messages sent between Aries SoCs residing on the same chassis [min, avg, max]
GD	Group Data	Data (in bytes) sent from a group [min, avg, max]
GM	Group Messages	Messages sent from a group [min, avg, max]
iGD	Intra-Group Data	Data (in bytes) sent between chassis of the same group [min, avg, max]
iGM	Intra-Group Messages	Messages sent between chassis of the same group [min, avg, max]
GGD	Group to Group Data	Data (in bytes) sent between two groups [max]
GGM	Group to Group Messages	Messages sent from a group [max]

Table 5 Parameter space for benchmark executions on Vilje and Piz Daint

Parameter	Vilje		Piz Daint	
n	8-256	8-256	8-128	8-256
ppn	1-16	1-16	1-8	1-8
l	16B-16MiB	16B-16KiB	16B-16MiB	16B-16KiB
PM	1-4	1-4	1-4	1-4
#executions	3	2	3	4
#points	9210		6912	

3.3 Benchmarking

Benchmarking is a core process in our methodology as it allows us to explore the potential relationship between the defined features and communication time, observe ubiquitous performance effects, and finally build communication models for the target system. Benchmarking supplies an incisive dataset to train predictive models. We devise our benchmark to resemble the communication phase of an application and to be parametric to various features, allowing us to sweep an ample space of communication configurations and traffic patterns. Note that the benchmarking step needs to be applied only once (e.g. after the initial deployment of the execution platform) and thus its complexity and execution overhead is not on the critical path.

```
for $l = 1$ to $max\_length$ do
    for $iter = 1$ to $Iterations$ do
        MPI_Barrier(MPI_COMM_WORLD)
        gettimeofday(start_time)
        for $m = 1$ to $ProcessMessages$ do
            MPI_Irecv(pong[$m$], $l$,
                MPI_UNSIGNED_CHAR, pair[$m$],...)
            MPI_Isend(ping[$m$], $l$,
                MPI_UNSIGNED_CHAR, pair[$m$],...)
        end for
        MPI_Waitall(2*$m$, ...)
        gettimeofday(stop_time)
        time[$iter$]=stop_time - start_time
    end for
    sort(time)
    local_reported_time=time[3*$Iterations$/4]
    MPI_Reduce(local_reported_time,
        global_reported_time, MPI_MAX...)
end for
```

Fig. 3 Benchmark core operations and timing

Our benchmark builds upon WICON [7], where random node pairs, with a single process hosted on each node, communicate in a ping-pong fashion simultaneously. WICON's goal is to quantify message latencies in the presence of contention. To capture contention and congestion effects due to multiple processes per node, we augmented the benchmark so that multiple processes hosted on each node communicate simultaneously with processes hosted on the paired node. To break the symmetry of this scheme, we switched from random node pairs to random MPI rank pairs, creating a fully randomized communication pattern. Moreover, to assess the impact of the number of messages sent by each process, we paired each process with M other randomly chosen processes, resulting in a scheme where each process sends M messages of the same length to M random processes and receives a reply. We also switched from blocking to non-blocking communication, as the latter allows overlapping at the system level between consecutive message transmissions and is a common practice for most MPI applications. The pseudocode for the core benchmark operations is listed in Figure 3.

Our benchmark allows us to sweep the parameter space explicitly for four class A features: the number of nodes (n), the number of processes per node (ppn), the message length (l) and the messages per process (PM). Implicitly, by sweep-

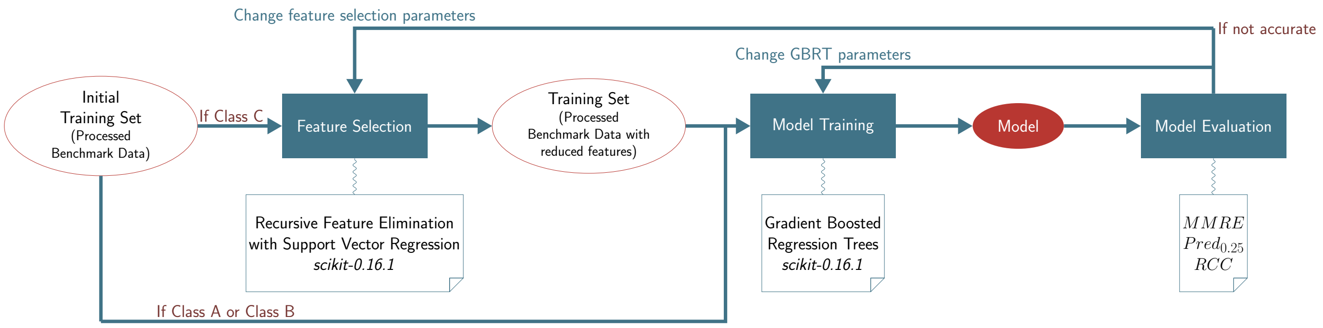

Change feature selection parameters
If not accurate
Change GBRT parameters
Initial Training Set (Processed Benchmark Data)
If Class C
Feature Selection
Training Set (Processed Benchmark Data with reduced features)
Model Training
Model
Model Evaluation
Recursive Feature Elimination with Support Vector Regression scikit-0.16.1
Gradient Boosted Regression Trees scikit-0.16.1
$MMRE$
$Pred_{0.25}$
RCC
If Class A or Class B

Fig. 4 Our model building process.

ing these four features, a large parameter space for all class A and B features is swept as well. Values of class C features are related to the allocation given to its job by the system schedulers, so there are two options for sweeping the parameter space for class C features: either to explicitly request specific node allocations with some specific configuration, or to execute the benchmark multiple times on different allocations, randomly picked by the scheduler. We followed the latter approach for both systems, as on Vilje, the former approach is not applicable. We performed three executions of the benchmark on both systems, as well as additional executions for short message lengths, for which we observed high variability for different allocations. The details of the benchmarking process are presented in Table 5. This dataset collected through benchmarking associates all the features with communication time and serves as the training set for the construction of the communication model on each system. The wall-clock time to collect the training set for each system is less than four hours.

To collect communication time measurements, the benchmark calls an *MPI_Barrier* before starting the timer (see Fig. 3), to avoid measuring process skew, (i.e. idle time), as communication time. We should note that although the *MPI_Barrier* is commonly utilized for process synchronization [33], its synchronization quality depends on the system and alternative synchronization methods can provide better measurement quality. Each process measures its local communication time for each iteration and reports the 3rd quantile of the times measured across all iterations. Through reduction, the root process reports the maximum of the locally reported times that designates the completion of the communication face. We chose the 3rd quantile of the local communication time as a good trade-off between the average and the maximum time among iterations, to avoid extreme outliers that could occur for a certain iteration and at the same time to take into account some discrepancies and unexpected increases in communication time due to noise, congestion or interference from nearby jobs.

3.4 Model building

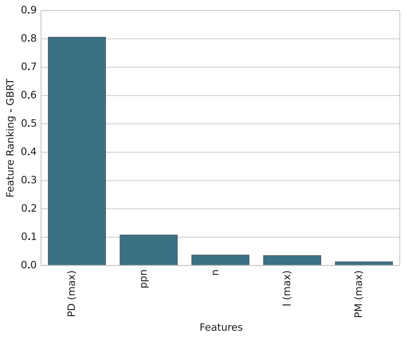
3.4.1 Ensemble methods

We consider several machine-learning techniques for regression ranging from multiple variable regression, to ensemble and boosting methods. We also experiment with several methods for feature selection to properly incorporate the most meaningful features out of the ones discussed in Section 3.2. For multiple variable regression, feature selection is inevitable, as it is part of the process to decide upon the final model form. For ensemble methods, feature selection is an optional pre-processing step, assisting in reducing training time, as it limits the set of available features.

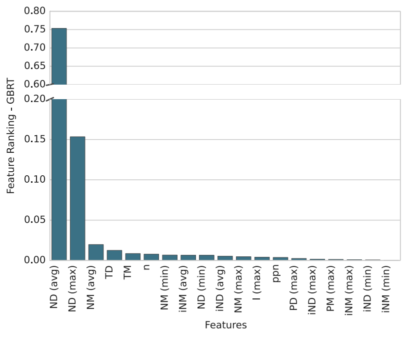
We prioritize ensemble methods such as Random Forests, Extra Randomized Trees, Gradient Boosting Regression Trees and AdaBoost Regression, all available in Python's *scikit-16.1* [46]. Automation of the model building process is the clear advantage of these methods which, in addition, eliminates the need to identify linear and non-linear relationships between the features and the target, interactions between features, high-order terms and parameter spaces that may more precisely describe the relationship between a feature and the target. Also, as ensemble methods combine multiple models, they improve in accuracy and robustness, compared to a single model[41]. These properties make ensemble methods an attractive solution for cross-system modeling, requiring no special effort to fit a model for each particular system. On the other hand, ensemble methods only provide a ranking of importance for the selected features and do not reveal the exact relationship between each feature and the target, communication time. Finally, we note that the training overhead was negligible as the training time itself was low (i.e. less than two minutes for the most time-consuming methods) and it needs to be applied only once. The modeling process is depicted in Figure 4.

3.4.2 Feature selection

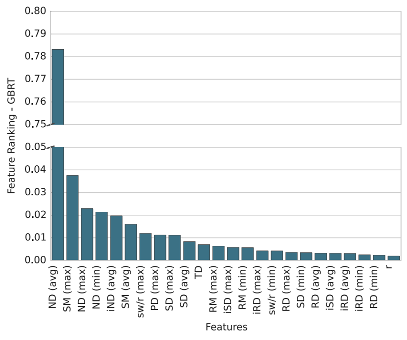
We start the model construction by feature selection on Class C features. Feature selection for Class A and Class B pre-



(a) *GBRT-Class A-Benchmark*

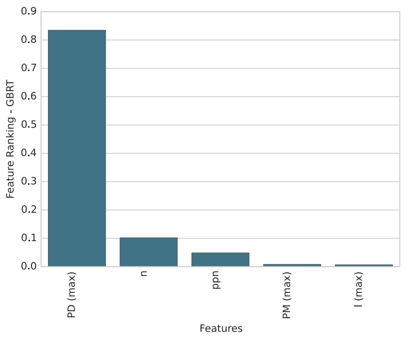


(b) *GBRT-Class B-Benchmark*

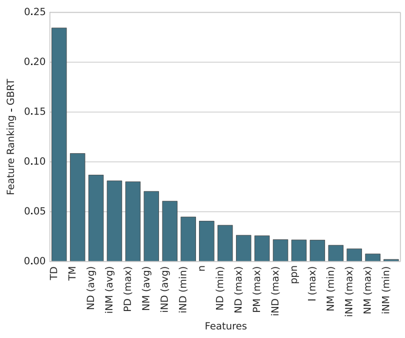


(c) *GBRT-Class C-Benchmark*

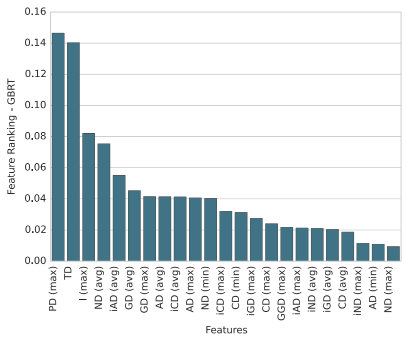
Fig. 5 Feature ranking for the benchmark-trained GBRT models on Vilje.



(a) *GBRT-Class A-Benchmark*



(b) *GBRT-Class B-Benchmark*



(c) *GBRT-Class C-Benchmark*

Fig. 6 Feature ranking for the benchmark-trained GBRT models on Piz Daint.

dictors is not necessary, as the default feature selection of the methods we apply is effective for the limited number of features of these classes. We rank Class C features using recursive feature elimination based on support vector regression, a method first proposed in [28] for genetic diagnostics. Support vector regression provides high generalization ability, while recursive feature elimination is important to avoid high ranking of features that overfit the training set. We experiment with feeding 15 to 40 of the highest-ranked features as inputs to the aforementioned methods. We select a similar number of features for both systems (24 features for Vilje and 26 for Piz Daint), as the size of the training set is also similar, yet selected features differ on the two systems. On Vilje, the selected features are more diverse and include features for the allocation shape (e.g. *r* and *sw/r*), while on Piz Daint all selected features are related to the data volume (e.g. *ND* and *AD*).

3.4.3 Modeling with Gradient Boosted Regression Trees

After feature selection, we experimented with the methods discussed before, and promoted Gradient Boosting Regression Trees (GBRT) as the most appropriate one for our problem and training set. GBRT is a boosting method, where a new weak learner, i.e. a regression decision tree, is iteratively fit on the negative gradient of a given loss function. GBRT offers high accuracy, alongside with automation and special tuning parameters that assist us in overcoming the perks of communication time prediction. We utilize the least absolute deviation as the loss function, which allows for robustness to outliers. This property of the loss function is critical for our task, as noise is present in measurements of communication time and can result in several outliers. To prevent our models from overfitting to the training set, we perform regularization by employing shrinkage, namely defining a learning rate by which the impact of each consecutive boosting iteration is shrunk, and by tuning the shape of the decision trees. To find the optimal configuration for the GBRT method and the optimal predictor, according to our training set, we test the method with different values for the learning rate (*learning_rate*) and the minimum samples per leaf (*min_samples_leaf*) of the decision trees and minimum samples at which a split is performed (*min_samples_split*), as well as different numbers of decision trees (*n_estimators*) and select the values that gave the model with the best fit, according to the evaluation metrics defined in Section 3.1. The range of values tested and the selected values for the parameters of the method for the two systems are presented in Table 6, along with the number of features (*features-classX*) utilized by the model for each class. We denote the models constructed with GBRT as *GBRT-Class A*, *GBRT-Class B* and *GBRT-Class C*, according to the class of features and the utilized training set.

Summarizing, to build a predictive model for communication time on a target system, we take the following actions: a) we define a number of features for the traffic pattern and the allocation shape and divide them in three classes, depending on their awareness to the mapping and the architecture, b) we benchmark the system once, using a single benchmark for point-to-point communication and sweeping a range of values for our features, c) we perform feature selection using recursive feature elimination and support vector regression, if the number of features is high enough to impede the modeling process, d) we feed the training set, collected from benchmarking, consisting of the selected features with their values and the corresponding measured communication time, to a model using the Gradient Boosted Regression Trees method, e) we iteratively tune the number of selected features and/or the parameters of the GBRT method, until we achieve high scores in the *Pred_0.25* and *RCC* metrics and a satisfactory range of relative errors.

3.4.4 Feature importance

GBRT do not supply a closed analytical form for communication time, however, *scikit-learn* provides a method to rank the features based on their impact to the output, on a scale from 0 to 1, allowing us to draw useful observations about factors that affect communication time. It is important to note, though, that this ranking is relative to the specific method and the training set and is not an accurate measure for the relevance of the various features to communication time: important features may have been ranked low or omitted, while highly-ranked features are usually utilized for splitting the dataset in higher levels of the decision trees but would not provide accurate predictions if not combined with the remaining features. Nevertheless, an overlook of the selected features (see Figures 5 and 6) can later explain the predictive ability of each model. For model *GBRT-Class A*, Process Data (*PD*) both on Vilje and Piz Daint, followed by the allocation size, as defined by *n* and *ppn*. For both systems, the method does not ignore the new features belonging to classes B and C when given as inputs, proving their correlation to communication time and validating our assumption that more information, extracted from the process mapping and the system architecture, can enhance the prediction ability of the method, as we will also demonstrate in Section 4. As for the highest-ranked features, for Vilje, Node Data (*ND*) is the predominant feature for both *GBRT-Class B* and *GBRT-Class C*, scoring higher than 0.7. On Piz Daint, equivalently, Total Data (*TD*) is the most highly ranked feature in class B, yet not dominant, as it scores less than 0.25. For class C, Process Data (*PD*) is the most highly ranked feature, although scoring less than 0.15. Overall, the feature ranking for *GBRT-Class C* on Piz Daint demonstrates that decision trees take into account the traffic through various points of the network at more levels in all depths of the trees, while on Vilje, metrics of traffic on the nodes and switches primarily affect the modeling process. However, on Vilje, the distribution of data to messages is also critical, as indicated by the importance of metrics for messages, which are pruned by feature selection on Piz Daint.

Table 6 Tested range of values and selected values for the parameters of the GBRT method and number of features

Parameter	**Range**	**Selected Value**	
		Vilje	Piz Daint
n_estimators	100 - 2000	500	1000
learning_rate	0.001 - 1	0.003	0.009
min_samples_leaf	1 - 5	2	2
min_samples_split	2 - 8	3	3
max_depth	3 - 8	7	5
features_classA	-	5	5
features_classB	-	19	19
features_classC	15 - 40	24	23

3.5 Alternative models

Although the multitude of factors affecting communication time advise for multiple variable models for communication time, we also detected features which were very highly correlated with communication time. This observation urges us to construct additional single-variable linear models for communication time. We worked towards this direction to test the importance of utilizing multiple features for communication prediction. As single variable models are very easy to build, we selected the most highly correlated feature for each system: *SD (max)* for Vilje and *AD (max)* for Piz Daint to build single variable regression models. It is noteworthy that these features, both belonging to Class C, are similar on the two systems, as they highlight traffic leaving the switching components (IB switch and Aries SoC). To identify the relationship between the selected feature and the target, and decide upon the model form, we constructed a vector of polynomial and non-linear transformations of the target feature, $X = \{1, x, x^2, x^3, log_2 x, log_2^2 x, log_2^3 x, \sqrt{x}\}, x \in \{SD(max), AD(max)\}$ and constructed 7806 linear models of the form $\hat{t} = b_0 + \sum_{i=1}^{3} b_i x_j x_k, x_j, x_k \in X, j \neq k$, trained with the benchmark data. We then selected the best-fitting model for each system, denoted as *LinReg - SV*, standing for single-variable linear regression. The models and their coefficients for the two systems are given in Table 3.5.

Table 7 Single Variable Linear Regression Models for Vilje and Piz Daint

System	Model Form	b_0	b_1	b_2	b_3
Vilje	$b_0 + b_1 \times log_2^5 x + b_2 \times xlog_2^2 x + b_3 \times x\sqrt{x}$	-5.153×10^{-4}	5.518×10^{-15}	5.651×10^{-11}	-1.471×10^{-16}
Piz Daint	$b_0 + b_1 \times \sqrt{x} + b_2 \times log_2^3 x + b_3 \times xlog_2^3 x$	7.939×10^{-6}	1.183×10^{-7}	-1.278×10^{-15}	5.151×10^{-15}

3.6 Extracting features from HPC applications

The features we define reflect the flows of data that result from the mapping of the communication graph to the network graph of a specific allocation. Features of different classes require different pieces of information, which can be extracted at different times in the execution lifetime of an application. Class A features only require information from the communication pattern, in the form of a time-independent trace, i.e. tuples of the form (*sender*, *receiver*, *message size*). This information can be automatically extracted with tracing tools, such as mpiP [53] or TAU [49] and *tau2simgrid* [17]. If the source code is available, this information can be extracted by simple code inspection. As we only need time-independent traces, i.e. we do not require time stamps for communication events, the collection of Class A features can be performed by tracing the application on fewer cores than the target scale, e.g. on a single or a few nodes. Class B features require information from the process mapping, i.e. the placement of processes on nodes, in the form of tuples (*rank*, *node*). The mapping is usually known to the user for an execution of an application, as the common practice is that the user defines either a default mapping for processes, such as block or round-robin placement of processes on nodes, or defines their own mapping, as a parameter to the *mpirun* wrapper script, or as an environment variable to Torque[5] or SLURM[6] scripts for job submission on HPC systems. Class C features require the list of nodes of the given allocation, which is given by the resource manager, Torque or SLURM, just before the execution of the application. They also require system-specific information from the underlying topology, which, in the case of InfiniBand networks, can be extracted by analyzing the output of *ibstat*[7] and in the case of Cray systems, it can be extracted by analyzing the output of *xtnodestat*[8]. In both cases, this output provides the position of any node on the underlying topology and allows the extraction of Class C features. It should be noted that the Select Plugin of SLURM takes the decision of node allocation after submission, thus Class C information can become available long before the execution of the application and not “just-in-time”.

5 http://www.adaptivecomputing.com/products/open-source/torque/
6 http://slurm.schedmd.com/
7 https://linux.die.net/man/8/ibstat
8 http://pubs.cray.com/#/Collaborate/00256453-FA

4 Evaluation

4.1 Communication patterns

We experimented with two communication patterns that are commonly encountered in real-world parallel applications, i.e. a *Halo-3D* exchange pattern, drawn from the 7-point-Jacobi relaxation, and a *Halo-4D* exchange pattern, drawn from Lattice QCD simulations. The *Halo-3D* (*Halo-4D*) exchange pattern is implemented with MPI as follows: processes are arranged in a virtual 3D (4D) cartesian grid and the original 3D (4D) domain is decomposed into smaller 3D subdomains (4D subdomains). Each process exchanges a 2D (3D)-face with each of the six (eight) neighboring processes.

We also applied our methodology to the point-to-point communication phases of LULESH [39], the Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics proxy application. LULESH is a stencil-based code in three dimensions and exposes three point-to-point communication patterns in each simulation time step. The first pattern, *LULESH-1*, is a 27-point 3D-halo exchange for the communication of force vectors, the second pattern, *LULESH-2*, is a 7-point 3D-halo exchange for the communication of artificial viscosity and the third pattern, *LULESH-3*, is a 27-point 3D-wavefront for the communication of positions and velocities.

The selected communication phases expose four diverse, but very common nearest-neighbor patterns with different communication characteristics. In *Halo-3D* and *LULESH-2*, communication consists of 6 messages of equal size, i.e. the six 2D-faces of the 3D-subdomain. This simple pattern, which frequently appears in applications, is also found in the LLNL AMG2013[9] and Kripke[10] proxy applications, the ExMatEx CoMD[11] proxy application, CloverLeaf3D, miniAMR and miniGhost of the Mantevo[12] MiniApp suite, MG and SP of the NAS[13] Parallel Benchmarks and the LBL ExaCT miniGMG[14] proxy application. In *Halo-4D*, communication is denser, consisting of 8 messages of equal size, i.e. the eight 3D-faces of the 4D-subdomain. This pattern is present in all quantum chromodynamic codes, as is tm-

9 https://codesign.llnl.gov/amg2013.php
10 https://codesign.llnl.gov/kripke.php
11 http://www.exmatex.org/comd.html
12 https://mantevo.org/
13 https://www.nas.nasa.gov/publications/npb.html
14 https://ccse.lbl.gov/ExaCT/index.html

Table 8 Details of the testing set

	Vilje			Piz Daint		
Pattern	Halo-3D	Halo-4D	LULESH	Halo-3D	Halo-4D	LULESH
Domain Size	128^3, 256^3, 512^3, 1024^3, 2048^3	128^4, 256^4	240^3, 480^3	128^3, 256^3, 512^3, 1024^3, 2048^3	128^4, 256^4	240^3, 480^3
Iterations	256	256	100	256	256	100
n	16-512	16-512	8-225	16-1024	16-1024	8-1000
ppn	1-16	1-16	1-16	1-8	1-8	1-8
#executions	3	2	1	3	2	1
#points	648			613		

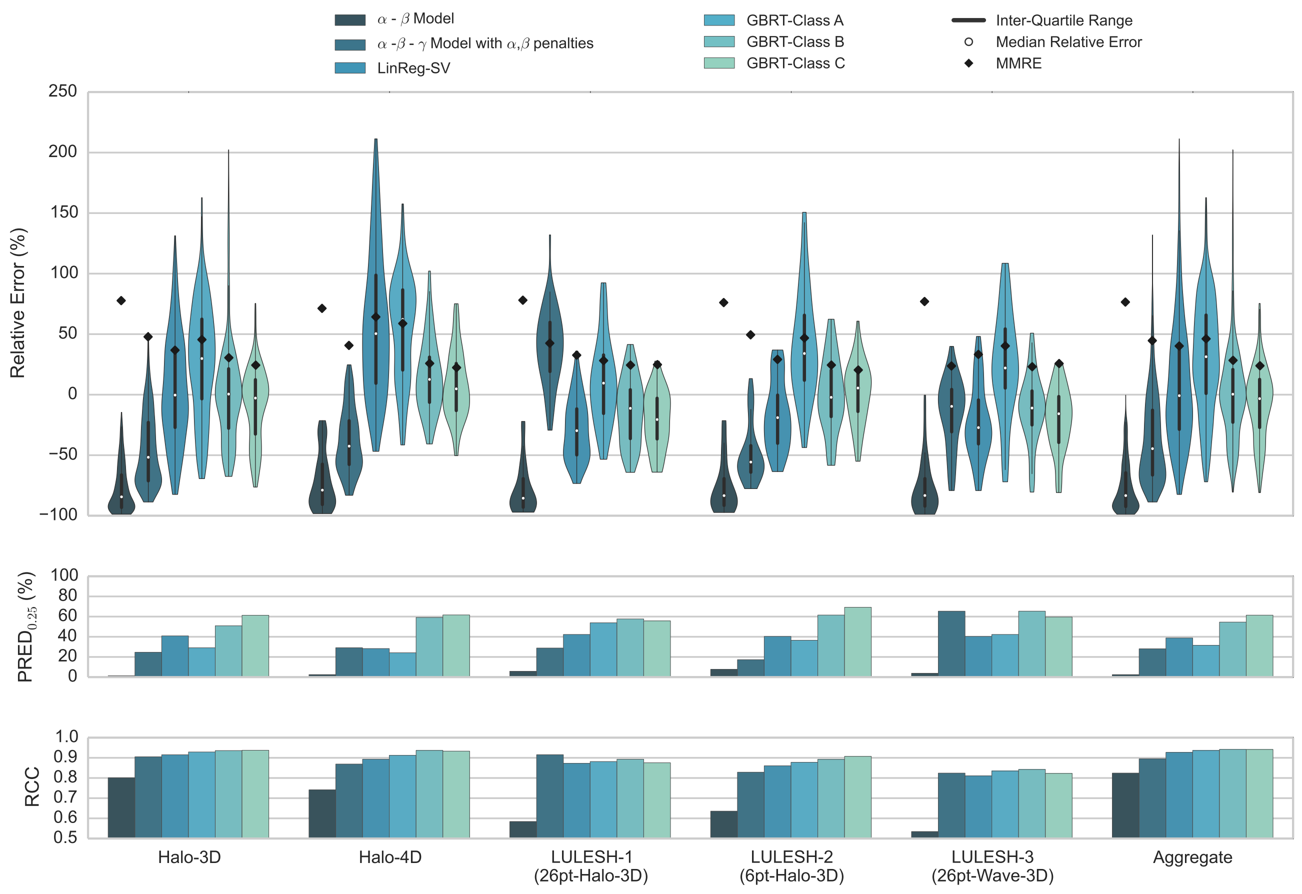


Fig. 7 Model comparison with relative error distribution, accuracy and goodness-of-fit metrics for Vilje.

LQCD[15], MILC[16] and PRACE QCD benchmarks[17]. *LULESH-1* processes exchange 26 messages of three different sizes, arising from the geometry of the 3D-subdomain: six 2D-faces, twelve 1D-edges and eight corners of one element. This pattern appears in numerous HPC applications and can be found in the LLNL Lassen[18] and AMG2013 proxy applications, the ANL CESAR NekBone[19] proxy application, as well as in HPCCG, miniFE and miniGhost of the Mantevo suite. *LULESH-3* exposes a wavefront pattern, in which processes exchange messages of three different sizes, namely faces, edges and corners, but communication takes place diagonally, i.e. each process sends only 13 messages to thirteen neighbors and receives 13 messages from the remaining thirteen neighbors.

We predict communication time for *Halo-3D*, *Halo-4D* and LULESH for various problem sizes and execution configurations, as well as for multiple executions, in order to test the ability of class C features to describe the effects of distinct allocations. The specifics of the testing set for the two systems is given in Table 8. LULESH by design expects

[15] https://github.com/etmc/tmLQCD
[16] http://physics.indiana.edu/ sg/milc.html
[17] http://www.prace-ri.eu/ueabs/#QCD
[18] https://codesign.llnl.gov/lassen.php
[19] https://cesar.mcs.anl.gov/content/software/thermal_hydraulics

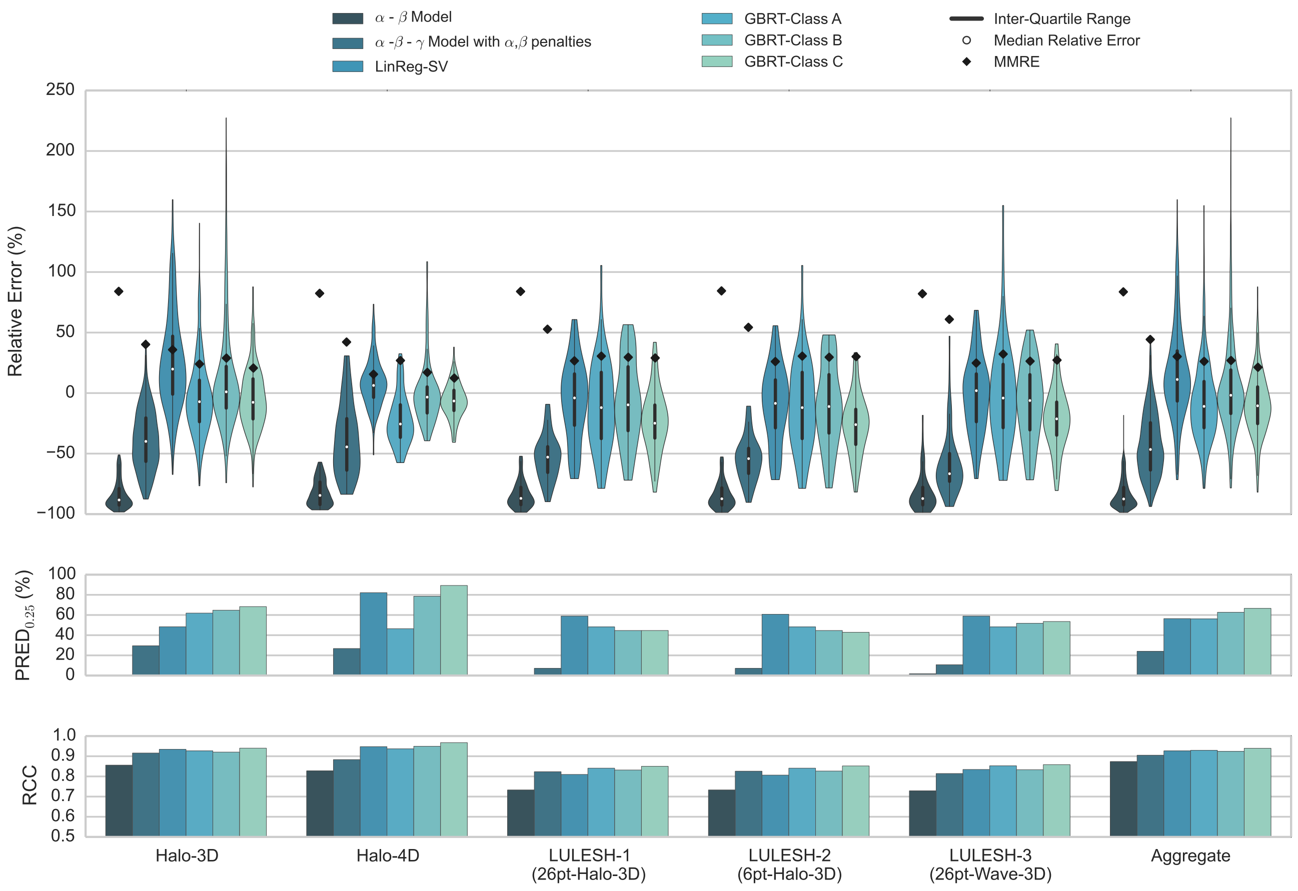


Fig. 8 Model comparison with relative error distribution, accuracy and goodness-of-fit metrics for Piz Daint.

Table 9 Measured parameters for the analytical and semi-empirical models on Vilje and Piz Daint

	Vilje	Piz Daint
α	0.305 us	0.238 us
β	0.215 ns	0.114 ns
γ	0.257 us	0.453 us

the number of processes to be a power of 3, so all execution configurations for LULESH are bound by this constraint.

4.2 Analytical and semi-empirical models

For comparison purposes, we implemented analytical and semi-empirical models for communication time, as proposed by Gahvari et al. [24, 25], for all the evaluated communication patterns. The baseline analytical approach is the $\alpha - \beta$ model, which is the equivalent of Hockney's latency-bandwidth model, where α is the latency and β is the inverse bandwidth. A third parameter, γ, denotes the per-hop delay and introduces a distance penalty, resulting in the α-β-γ model. We measured the three parameters with the HPCC benchmark[20] on Vilje and Piz Daint and show their values in Table 4.2. The models can be enhanced with three penalties. A penalty can be added to the β parameter, to reflect the limits of achievable bandwidth and network contention due to link sharing. A second penalty can be added to the α parameter, to capture the effect of multiple processes accessing the network from the same node (multicore penalty). The same penalty can be added to the γ parameter, as multiple processes can create contention on every hop of a message on the network. The addition of these penalties result to 5 possible models. We evaluated all five models and present prediction results for the analytical α-β model and the best of the five semi-empirical, penalized models on each system, which is the α-β-γ model with α, β penalties. Note that, while we do not expect the α-β model to capture the complexities of communication time and give accurate prediction results, it constitutes the simplest expression for communication time, captures the effect of the problem size and decomposition and thus is a lower bound for prediction accuracy.

[20] http://icl.cs.utk.edu/hpcc/

4.3 Model comparison

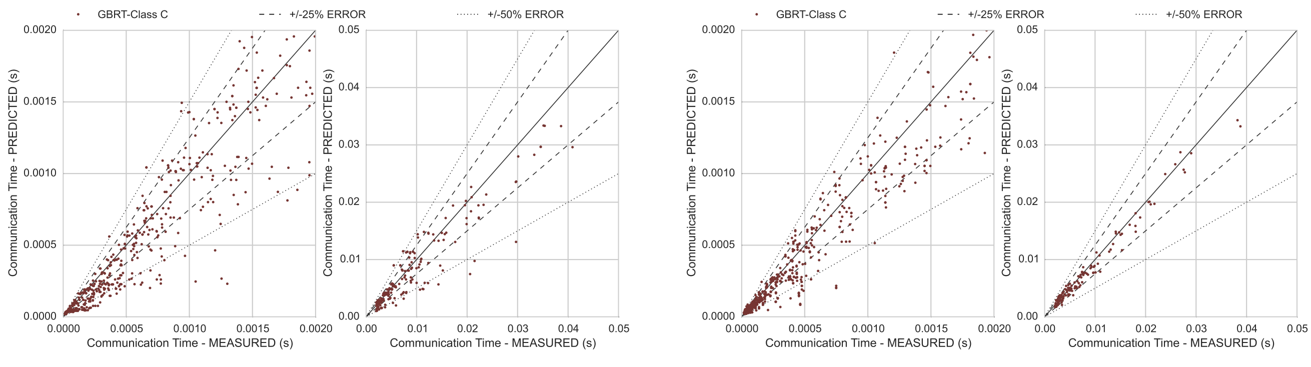
To evaluate the predictive ability of our models, we utilize the metrics defined in Section 3.1. Figures 7 and 8 show the comparison of the six types of models described in Section 3 and Section 4.2. We may draw a number of interesting observations regarding the modeling methodology. First, adding empirical knowledge, mapping and architecture awareness to prediction models significantly improves accuracy. Our empirical models outperform the semi-empirical penalized α-β-γ model in all cases. On aggregate, with our *GBRT-Class C* we achieve to reduce the *MMRE* from 44.64% of the penalized model to 23.98% on Vilje and from 42.6% to 21.31% on Piz Daint. Second, models trained with benchmark data are effective in predicting communication time. This validates our choice to rely on generic, application-independent benchmarking, which requires data collection once, at the initialization of the modeling process. Third, single variable regression models can be accurate enough when their single feature is capable of capturing core effects of actual communication. However, they cannot generally provide high accuracy across platforms and applications. For example, on Vilje, the selected feature reflects maximum application traffic through a switch (*SD (max)*), which in general does not correspond to the actual traffic on the switch: switches are often shared among different allocations and applications. Finally, the baseline α-β model fails to predict communication time in all tested patterns, although it manages to capture part of the scaling behavior of communication, owed to the problem size and decomposition, as reflected in its *RCC* scores.

Focusing on Vilje, the $LinReg - SV$ model exhibits the lowest accuracy among our models, however, looking at its high *RCC* score, it is able to distinguish between different communication configurations. In this system, Class C models clearly outperform all other approaches. The prediction results of the penalized α-β-γ model are interesting: while it generally underpredicts most patterns, it overpredicts *LULESH-1* and performs well in *LULESH-3*. This is a result of the penalty on β, where the ratio of total number of messages to number of links is taken into account. The two patterns exhibit high values for the number of messages. On Piz Daint, the Class C model exhibits the highest aggregate accuracy across the various communication patterns. Interestingly, model *LinReg - SV* achieves remarkable accuracy, especially in *LULESH* and *Halo-4D* patterns, as the selected feature, *AD (max)*, that corresponds to the traffic through an Aries SoC, highly characterizes the traffic pattern and communication performance. Finally, an observation for *GBRT-Class A* is that, while the model scores decently on Piz Daint, it often overpredicts communication time on Vilje for all patterns. This behavior proves that Class A features, in combination with our generic benchmarking, are insufficient for sketching the traffic pattern and achieving accurate predictions on Vilje, where communication performance is highly influenced by effects occurring at the node and switch level and the distribution of data to different messages is important, as indicated by the feature ranking for Class B and Class C models. The accuracy of *GBRT-Class A* can be boosted by augmenting the training set with data from irregular patterns, in order to include minimum and average values for the features *l*, *PD* and *PM*.

4.4 Detailed prediction evaluation

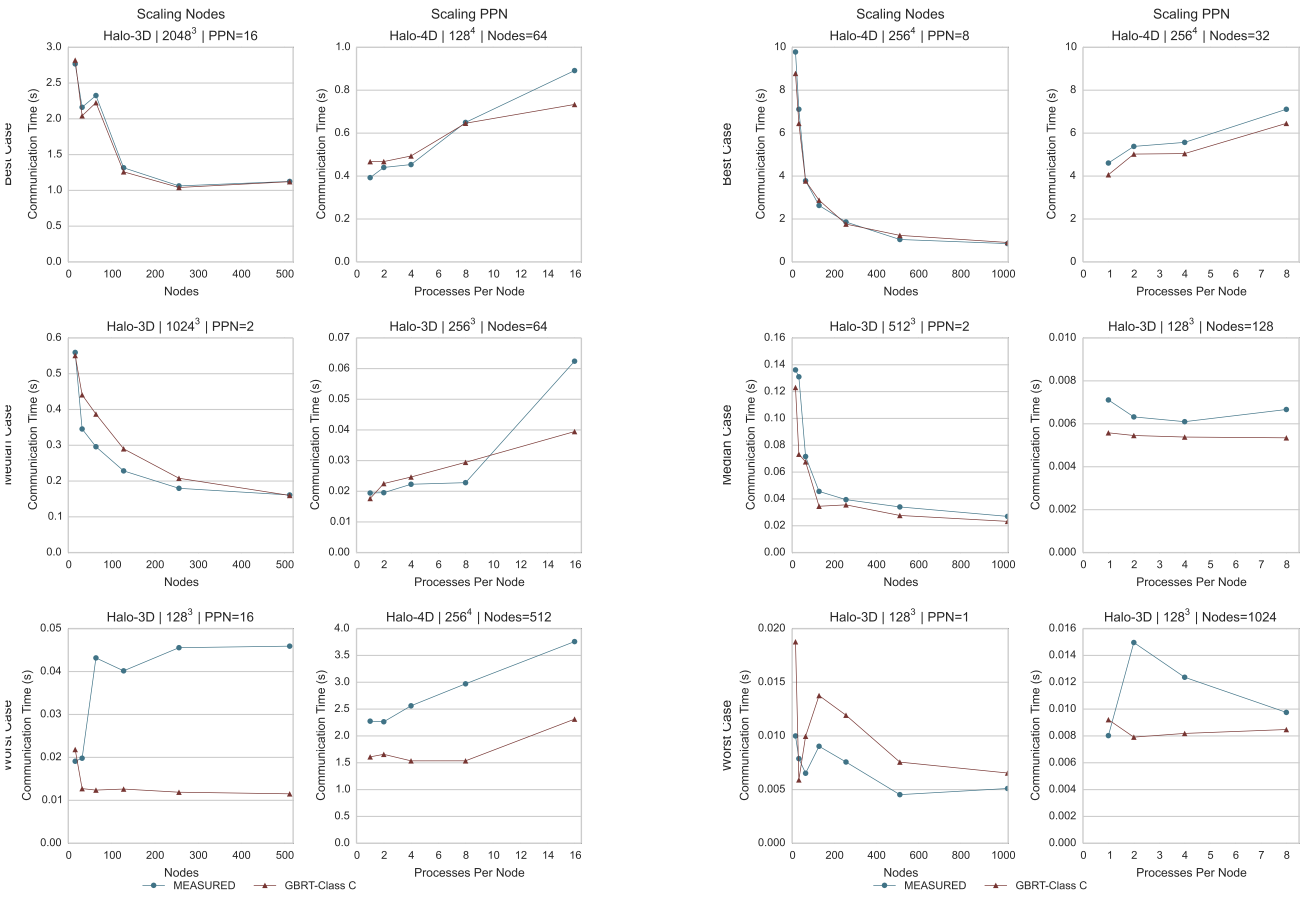
Based on the remarks of the previous paragraph, the best model across both systems and datasets is *GBRT-Class C*. We consider this model to be the most useful, as a) it is applicable just-in-time before the execution of the application (whereas Class A and Class B models which have better applicability are not equally successful for all communication patterns), b) its accuracy is on aggregate very high across both systems, scoring 61.43% in $Pred_{0.25}$ on Vilje and 66.57% on Piz Daint and c) its goodness-of-fit is excellent, as its *RCC* score is 0.942 on Vilje and 0.940 on Piz Daint. For this reason, we further analyze the accuracy of this model. Fig. 9 presents a comparison of the measured and predicted values with *GBRT-Class C*, for all points in the testing set, normalized to one iteration, in the form of scatterplots, broken down by communication time for the sake of visibility. On Vilje (see 9(a)), the majority of predictions (61.43%) lie within the $\pm$25% error line, while 87.6% of predictions lie within the $\pm$50% error line. A set of 60 points with communication time lower than 0.0015 seconds lie below the -50% error line. These points correspond to configurations with short message lengths and high core counts, where communication time measurements are noisy and exhibit high time variability, hence our model tends to underpredict for these configurations. The scatterplots for Piz Daint in Fig. 9(b) indicate that there are no systematic under- or over-predictions on this system. The model is well-crafted for any communication configuration and 92.14% of predictions lie within the $\pm$50% error line, with very few outliers present when measured communication time is lower than 100 microseconds.

For the *Halo-3D* and *Halo-4D* patterns, our testing set includes a multitude of configurations, problem sizes and distinct executions on different allocations. To test the ability of our model to predict communication time scalability when the number of nodes or processes per node is scaled, we utilized the average of $Pred_{0.25}$ and *RCC* to score the various subsets of measurements. We present the best, median and worst predicted subsets in Fig. 10. On Vilje (see Fig. 10(a)), predictions are excellent in the best and median case, both when scaling nodes and when scaling processes per node. This observation proves the ability of our



(a) Vilje (b) Piz Daint

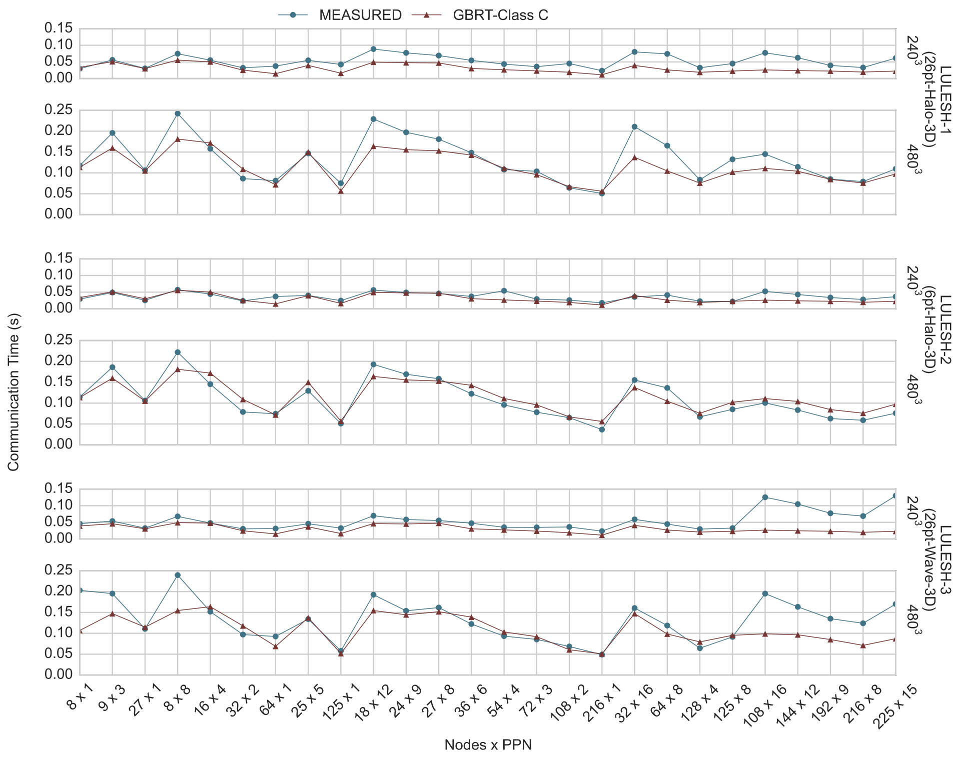
Fig. 9 Scatterplots for predictions with *GBRT-Class C*. Communication time is normalized to a single iteration.



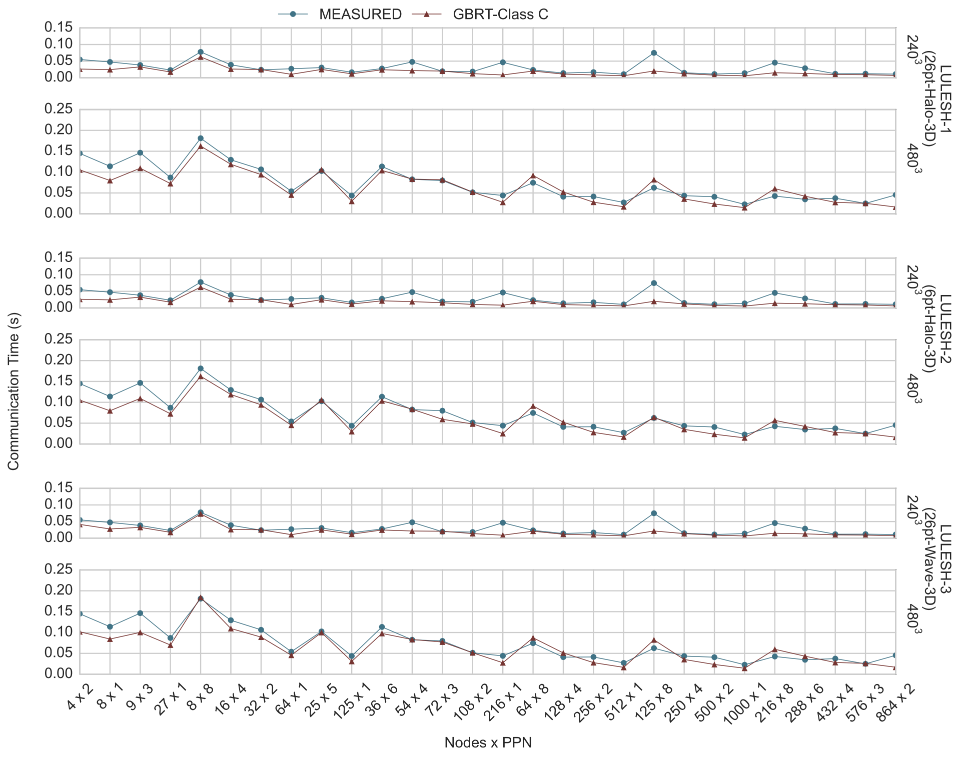
(a) Vilje (b) Piz Daint

Fig. 10 Predictions for Halo-3D and Halo-4D with *GBRT-Class C*. Best, median and worst cases are selected using the best, median and worst value of $Pred_{0.25} * RCC$.

model to achieve high prediction accuracy in most cases. The worst case presented, when nodes are scaled, corresponds to a small problem size with short messages, where noisy measurements and outliers are common and accurate predictions cannot easily be obtained, resulting in higher prediction errors. However, when processes per node are scaled, even in the worst case, communication scalability is well predicted. On Piz Daint (see Fig. 10(b)), we observe that our model achieves high quality predictions in the best and median case. Communication scalability is not accurately captured only in the worst case, where the problem size is also small with short messages.



(a) Vilje.



(b) Piz Daint.

Fig. 11 Predictions for LULESH with *GBRT-Class C*.

In Fig. 11, we present the predicted communication time for all executed configurations of LULESH, sorted by the number of cores. For all three patterns and two problem sizes in LULESH on Vilje (see Fig. 11(a)), communication time is predicted with high accuracy up to 216 cores (216 x 1). Configurations up to 512 cores (128 x 4) are also well distinguished by our model. In the case of *LULESH-3*, we observe under-predicted communication times for more than 512 cores. We should note that *LULESH-3* has similar communication volume with *LULESH-1*, sourcing from a different communication pattern: processes in *LULESH-3* exchange roughly half the number of messages compared to *LULESH-1*, with double length; yet communication time rises in *LULESH-3* for more than 512 cores. This behavior is inconsistent with observations in our training set, which explains the shortcoming of the model. On Piz Daint (see Fig. 11(b)), accurate predictions are accomplished for the majority of configurations, with the exception of four configurations (54 x 4, 216 x 1, 125 x 8, 216 x 8) where communication time is underpredicted in all three patterns for the small problem size (240^3) and two configurations (8 x 1, 864 x 2) where communication time is underpredicted for the large problem size (480^3), though these configurations are dissimilar and mispredictions may be due to noise or interference with other applications at the time of execution.

5 Related work

Analytical performance modeling of the communication time of parallel applications has been a hot topic in the past two decades. Hockney's model [29] for point-to-point communication was a meaningful approach to express communication time for a pair of processors as a function of network-related parameters, namely startup time and maximal bandwidth. Culler et al. [15] attempted an elaborate specification of end-to-end communication time based upon network properties and the message or packet length, giving birth to the Log(G)P model family. LogGP [2] and LogGPS [35] extend LogP with parameters for different message sizes and have been state-of-art in communication performance modeling, however they come with several weaknesses and restrictions. Their major weakness is their focus on a local instance of the network, disregarding global network effects, such as overlapping, multiple hops, contention or congestion, which are addressed separately in LogGP extensions (LoGPC [42], LoPC [22], LoGPG [43]). As novel network architectures with offload-enabled network interfaces have emerged, models of the LogGP family are no longer applicable [18]; new parameters need to be introduced to capture operations on the network interfaces. In addition, building a model for a real-life application under Log(G)P parameters is not straightforward and several works, [44, 4] deploy the LogGP model to build communication performance models for large-scale applications on specific machines. More recent works [5, 56] attempt to encapsulate network contention in novel network models, limiting to specific interconnection network architectures, still restraining, though, to the modeling of network characteristics and communication primitives, as do the models of the Log(G)P family. Overall, analytical models trade accuracy for low awareness and high applicability through simplicity [32]. Considering their purpose, analytical models are meaningful for hardware/software co-design and communication software design.

The alternative to analytical modeling is empirical modeling, i.e. the utilization of measurements on the target system for the modeling and prediction of application performance, through benchmarking or information collection at runtime. Empirical models, contrary to analytical ones, can achieve high accuracy by incorporating high levels of awareness, sacrificing part of their applicability. Related work on empirical modeling includes the work of Jain et al. [37] and Bhatele et al. [9], where performance models for communication are built for Blue Gene/Q deploying the network's performance counters. These models are highly accurate, as performance counter measurements allow for full awareness of the traffic pattern and runtime conditions on the given allocation, yet their applicability is limited: the values of the features can only be observed after the execution of an application, not allowing for communication time prediction ahead of execution. In addition, their methodology is strictly limited to systems as the BlueGene/Q, where network components in use are dedicated to the allocation for the application execution and performance counter measurements correspond only to traffic generated from the application in study. A semi-empirical approach is taken by Gahvari et al. [24, 25], where the latency-bandwidth model is extended with empirical parameters for the modeling of the Algebraic Multigrid. This approach enhances the accuracy of the latency-bandwidth model by adding awareness of the process mapping, network architecture and topology, and is applicable at runtime prior to the execution, for many applications that expose point-to-point communication patterns, as the Fast Multipole Method [34]. Empirical models increase prediction accuracy and are meaningful for enhancing decisions taken at runtime, e.g. scheduling decisions and application tuning at runtime. Their drawback is that they require measurements on the target system.

Our work proposes a methodology for empirical predictive modeling of communication time, portable to any platform, by defining meaningful features for communication performance, available just-in-time ahead of execution. We utilize benchmarking to train machine-learning prediction models for the communication time of parallel applications on the target system.

6 Conclusions and future work

In this work, we presented a methodology to construct predictive models for the communication time of HPC applications. Following an empirical approach to predictive modeling, we defined features for communication performance on two systems, Vilje and Piz Daint, collected a set of measurements with benchmarking and constructed four predictive models for communication time, with different machine - learning methods and features. We evaluated the predictive ability of our models on halo-exchange communication patterns and LULESH, and conclude that multiple well-defined features, generic benchmarking and automated machine - learning methods, as gradient-boosted regression trees, offer a predictive model for point-to-point communication patterns with high accuracy and good applicability across all tested communication patterns and execution configurations. To the best of our knowledge, this is the first work that presents a cross-application, cross-platform methodology for communication time prediction of HPC applications ahead of execution, able to deliver high accuracy and goodness-of-fit, as indicated by our $MMRE$, RCC and $Pred_{0.25}$ scores, which reach 23.98%, 0.942 and 61.43% on Vilje and 21.31%, 0.940 and 66.56% on Piz Daint.

In future work, we intend to extend our methodology to target irregular communication patterns and collective communication. To capture irregular communication patterns, where processes exchange different number of messages of different sizes (as in iterative solvers involving sparse matrix computations), we need to extend our benchmarking step with irregular communication and take into account minimum, average and maximum values for Class A features (i.e. the message length l, the process data PD and process messages PM). Collective communication modeling shares many of the challenges with point-to-point communication modeling, however the communication pattern is predetermined. As such, we anticipate that some regular collective patterns can be well-modeled with a single or a few features, e.g. the number of processes, as indicated by the work of Shudler et al. [50], while other collective patterns can be modeled as point-to-point communication, since MPI implementations also internally use point-to-point communication for several irregular and many-to-many collectives. Finally, we aim to enhance our methodology with additional benchmarking data, collected under the presence of computation / communication overlapping, in order to capture communication behavior in this case as well.

Acknowledgements This research was supported in part with computational resources at the Norwegian Institute of Science and Technology (NTNU) provided by NOTUR and in part by a grant of computational resources from the Swiss National Supercomputing Center (CSCS) under project ID g83. Nikela Papadopoulou has received funding from IKY fellowships of excellence for postgraduate studies in Greece - SIEMENS program. The authors would also like to thank Sotirios Apostolakis for his contribution to the early steps of this work.

References

1. Alawneh L, Hamou-Lhadj A, Hassine J (2016) Segmenting large traces of inter-process communication with a focus on high performance computing systems. Journal of Systems and Software 120:1–16
2. Alexandrov A, Ionescu MF, Schauser KE, Scheiman C (1995) 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, ACM, pp 95–105
3. Alverson B, Froese E, Kaplan L, Roweth D (2012) Cray xc series network. Cray Inc, White Paper WP-Aries01-1112
4. Bauer G, Gottlieb S, Hoefler T (2012) Performance modeling and comparative analysis of the MILC lattice QCD application su3_rmd. In: Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on, IEEE, pp 652–659
5. Bédaride P, Degomme A, Genaud S, Legrand A, Markomanolis G, Quinson M, Stillwell ML, Suter F, Videau B, et al (2013) Toward better simulation of MPI applications on Ethernet/TCP networks. In: PMBS13-4th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems
6. Bekas C, Curioni A (2010) A new energy aware performance metric. Computer Science-Research and Development 25(3-4):187–195
7. Bhatelé A, Kalé LV (2009) Quantifying network contention on large parallel machines. Parallel Processing Letters 19(04):553–572
8. Bhatele A, Mohror K, Langer SH, Isaacs KE (2013) There goes the neighborhood: performance degradation due to nearby jobs. In: Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, p 41
9. Bhatele A, Titus AR, Thiagarajan JJ, Jain N, Gamblin T, Bremer PT, Schulz M, Kale LV (2015) Identifying the culprits behind network congestion. In: Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International, IEEE, pp 113–122
10. Böhme D, Geimer M, Arnold L, Voigtlaender F, Wolf F (2016) Identifying the root causes of wait states in large-scale parallel applications. ACM Transactions on Parallel Computing (TOPC) 3(2):11
11. Casanova H, Desprez F, Markomanolis GS, Suter F (2015) Simulation of mpi applications with time-

independent traces. Concurrency and Computation: Practice and Experience 27(5):1145–1168

12. Chandrashekar G, Sahin F (2014) A survey on feature selection methods. Computers & Electrical Engineering 40(1):16–28
13. Chen W, Zhai J, Zhang J, Zheng W (2009) LogGPO: An accurate communication model for performance prediction of MPI programs. Science in China Series F: Information Sciences 52(10):1785–1791
14. Conte SD, Dunsmore HE, Shen VY (1986) Software engineering metrics and models. Benjamin-Cummings Publishing Co., Inc.
15. Culler D, Karp R, Patterson D, Sahay A, Schauser KE, Santos E, Subramonian R, Von Eicken T (1993) LogP: Towards a realistic model of parallel computation, vol 28. ACM
16. Demmel J, Hoemmen M, Mohiyuddin M, Yelick K (2008) Avoiding communication in sparse matrix computations. In: Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on, IEEE, pp 1–12
17. Desprez F, Markomanolis GS, Quinson M, Suter F (2011) Assessing the performance of mpi applications through time-independent trace replay. In: 2011 40th International Conference on Parallel Processing Workshops, IEEE, pp 467–476
18. Di Girolamo S, Jolivet P, Underwood KD, Hoefler T (2015) Exploiting offload enabled network interfaces. In: High-Performance Interconnects (HOTI), 2015 IEEE 23rd Annual Symposium on, IEEE, pp 26–33
19. Drosinos N, Koziris N (2004) Performance comparison of pure mpi vs hybrid mpi-openmp parallelization models on smp clusters. In: Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International, IEEE, p 15
20. Ferreira KB, Bridges PG, Brightwell R, Pedretti KT (2013) The impact of system design parameters on application noise sensitivity. Cluster computing 16(1):117–129
21. Filgueira R, Singh DE, Carretero J, Calderón A, García F (2011) Adaptive-CoMPI: Enhancing MPI-based applications' performance and scalability by using adaptive compression. International Journal of High Performance Computing Applications 25(1):93–114
22. Frank MI, Agarwal A, Vernon MK (1997) LoPC: modeling contention in parallel algorithms, vol 32. ACM
23. Friedman J, Hastie T, Tibshirani R (2009) The elements of statistical learning: Data mining, inference, and prediction. Springer Series in Statistics
24. Gahvari H, Baker AH, Schulz M, Yang UM, Jordan KE, Gropp W (2011) Modeling the performance of an algebraic multigrid cycle on hpc platforms. In: Proceedings of the international conference on Supercomputing, ACM, pp 172–181
25. Gahvari H, Gropp W, Jordan KE, Schulz M, Yang UM (2014) Algebraic multigrid on a dragonfly network: First experiences on a cray xc30. In: International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems, Springer, pp 3–23
26. Goumas G, Sotiropoulos A, Koziris N (2001) Minimizing completion time for loop tiling with computation and communication overlapping. In: Parallel and Distributed Processing Symposium., Proceedings 15th International, IEEE, pp 10–pp
27. Goumas G, Drosinos N, Koziris N (2009) Communication-aware supernode shape. Parallel and Distributed Systems, IEEE Transactions on 20(4):498–511
28. Guyon I, Weston J, Barnhill S, Vapnik V (2002) Gene selection for cancer classification using support vector machines. Machine learning 46(1-3):389–422
29. Hockney RW (1994) The communication challenge for MPP: Intel Paragon and Meiko CS-2. Parallel computing 20(3):389–398
30. Hoefler T, Schneider T, Lumsdaine A (2010) Characterizing the influence of system noise on large-scale applications by simulation. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE Computer Society, pp 1–11
31. Hoefler T, Schneider T, Lumsdaine A (2010) LogGOPSim: simulating large-scale applications in the LogGOPS model. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, ACM, pp 597–604
32. Hoefler T, Gropp W, Kramer W, Snir M (2011) Performance modeling for systematic performance tuning. In: State of the Practice Reports, ACM, p 6
33. Hunold S, Carpen-Amarie A (2015) On the impact of synchronizing clocks and processes on benchmarking MPI collectives. In: EuroMPI, ACM, pp 8:1–8:10
34. Ibeid H, Yokota R, Keyes D (2016) A performance model for the communication in fast multipole methods on high-performance computing platforms. International Journal of High Performance Computing Applications p 1094342016634819
35. Ino F, Fujimoto N, Hagihara K (2001) LogGPS: a parallel computational model for synchronization analysis. In: ACM SIGPLAN Notices, ACM, vol 36, pp 133–142
36. Isaacs K, Gamblin T, Bhatele A, Schulz M, Hamann B, Bremer P (2014) Ordering traces logically to identify lateness in parallel programs. Tech. rep., Technical Report LLNL-TR-656141, Lawrence Livermore National Laboratory

37. Jain N, Bhatele A, Robson MP, Gamblin T, Kale LV (2013) Predicting application performance using supervised learning on communication features. In: Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, p 95
38. Jokanovic A, Sancho JC, Rodríguez G, Lucero A, Minkenberg C, Labarta J (2015) Quiet neighborhoods: Key to protect job performance predictability. 29th IEEE International Parallel & Distributed Processing Symposium (IPDPS-2015)
39. Karlin I, Bhatele A, Chamberlain BL, Cohen J, Devito Z, Gokhale M, Haque R, Hornung R, Keasler J, Laney D, Luke E, Lloyd S, McGraw J, Neely R, Richards D, Schulz M, Still CH, Wang F, Wong D (2012) Lulesh programming model and performance ports overview. Tech. Rep. LLNL-TR-608824
40. Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, Czajkowski G (2010) Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, ACM, pp 135–146
41. Mendes-Moreira J, Soares C, Jorge AM, Sousa JFD (2012) Ensemble approaches for regression: A survey. ACM Computing Surveys (CSUR) 45(1):10
42. Moritz CA, Frank MI (1998) LoGPC: Modeling network contention in message-passing programs. ACM SIGMETRICS Performance Evaluation Review 26(1):254–263
43. Moritz CA, Frank MI (2001) LoGPG: Modeling network contention in message-passing programs. Parallel and Distributed Systems, IEEE Transactions on 12(4):404–415
44. Mudalige GR, Vernon MK, Jarvis SA (2008) A plug-and-play model for evaluating wavefront computations on parallel architectures. In: Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on, IEEE, pp 1–14
45. Papadopoulou N, Goumas GI, Koziris N (2015) A machine-learning approach for communication prediction of large-scale applications. In: 2015 IEEE International Conference on Cluster Computing, CLUSTER 2015, Chicago, IL, USA, September 8-11, 2015, pp 120–123
46. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12:2825–2830
47. Rabenseifner R, Hager G, Jost G (2009) Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes. In: Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on, IEEE, pp 427–436
48. Sancho JC, Barker KJ, Kerbyson DJ, Davis K (2006) Quantifying the potential benefit of overlapping communication and computation in large-scale scientific applications. In: SC 2006 Conference, Proceedings of the ACM/IEEE, IEEE, pp 17–17
49. Shende SS, Malony AD (2006) The tau parallel performance system. International Journal of High Performance Computing Applications 20(2):287–311
50. Shudler S, Calotoiu A, Hoefler T, Strube A, Wolf F (2015) Exascaling your library: Will your implementation meet your expectations? In: Proceedings of the 29th ACM on International Conference on Supercomputing, ACM, pp 165–175
51. Tallent NR, Adhianto L, Mellor-Crummey JM (2010) Scalable identification of load imbalance in parallel executions using call path profiles. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE Computer Society, pp 1–11
52. Valiant LG (1990) A bridging model for parallel computation. Communications of the ACM 33(8):103–111
53. Vetter J, Chambreau C (2005) mpip: Lightweight, scalable mpi profiling
54. Yu L, Li D, Mittal S, Vetter JS (2014) Quantitatively modeling application resilience with the data vulnerability factor. In: High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for, IEEE, pp 695–706
55. Zhang C, Ma Y (2012) Ensemble machine learning. Springer
56. Zhu J, Lastovetsky A, Ali S, Riesen R, Hasanov K (2015) Asymmetric communication models for resource-constrained hierarchical ethernet networks. Concurrency and Computation: Practice and Experience 27(6):1575–1590