A Survey of Communication Performance Models for High-Performance Computing

JUAN A. RICO-GALLEGO and JUAN C. DÍAZ-MARTÍN, University of Extremadura, Spain
RAVI REDDY MANUMACHU and ALEXEY L. LASTOVETSKY,
University College Dublin, Ireland

This survey aims to present the state of the art in analytic communication performance models, providing sufficiently detailed descriptions of particularly noteworthy efforts. Modeling the cost of communications in computer clusters is an important and challenging problem. It provides insights into the design of the communication pattern of parallel scientific applications and mathematical kernels and sets a clear ground for optimization of their deployment in the increasingly complex high-performance computing infrastructure. The survey provides background information on how different performance models represent the underlying platform and shows the evolution of these models over time from early clusters of single-core processors to present-day multi-core and heterogeneous platforms. Prospective directions for future research in the area of analytic communication performance modeling conclude the survey.

CCS Concepts: • **Computing methodologies** → **Modeling methodologies**; *Parallel computing methodologies*; • **Networks** → *Network performance modeling*;

Additional Key Words and Phrases: Communication performance models, analytic modeling, communication performance, high-performance computing

ACM Reference format:
Juan A. Rico-Gallego, Juan C. Díaz-Martín, Ravi Reddy Manumachu, and Alexey L. Lastovetsky. 2019. A Survey of Communication Performance Models for High-Performance Computing. *ACM Comput. Surv.* 51, 6, Article 126 (January 2019), 36 pages.
https://doi.org/10.1145/3284358

1 INTRODUCTION

According to Rothenberg (Rothenberg et al. 1989), "Modeling, in the broadest sense, is the cost-effective use of something in place of something else for some cognitive purpose. It allows us to use something that is simpler, safer or cheaper than reality instead of reality for some purpose. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and

This work was supported by the European Regional Development Fund "A way to achieve Europe" (ERDF) and the Extremadura Local Government (Ref. IB16118) and the Science Foundation Ireland (SFI) under Grant Number 14/IA/2474. This work was partially supported by the EU under the COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS).
Authors' addresses: J. A. Rico-Gallego, University of Extremadura, Escuela Politécnica, Avd. Universidad s/n, Cáceres, 10003, Spain; email: jarico@unex.es; J. C. Díaz-Martín, University of Extremadura, Cáceres, Spain; email: juancarl@unex.es; R. R. Manumachu and A. L. Lastovetsky, University College Dublin, Belfield, Dublin 4, Dublin, Ireland; emails: {ravi.manumachu, alexey.lastovetsky}@ucd.ie.

0360-0300/2019/01-ART126 $15.00
https://doi.org/10.1145/3284358

irreversibility of reality." The current reality of scientific programming is indeed complex. Modern high-performance computing (HPC) platforms present fast networks linking deep memory hierarchies culminated by multi- and many-core GPPs and accelerators, with the goal of reaching the highest possible performance. This survey is about *Communication Performance Models* for HPC. Today's HPC is mostly done in clusters. The 51st TOP500 list (Top500 2018), published in June 2018, sets that 437 out of 500 entries are clusters. The HPC industry relies on software tools for the intriguing task of determining the performance gain or loss of using a given component, either hardware or software, but using formal prediction techniques is the way to make such analysis easier and probably scalable. Performance prediction remains a significant unmet challenge in the area of HPC.

Communication performance modeling now faces the challenge of the increasing complexity of the computing nodes of current networks. It is with good reason that the so named *node-level performance engineering* is now a well-established branch of computer engineering. The cluster node has become multi-core. Processes running in the same node usually communicate through a buffer in shared memory, but direct transmissions are also possible using operating system modules, operating system bypass or Remote Memory Access (RMA) through high-performance networks such as *Infiniband*. Of course, performance models have to deal with these and other issues, some of which are hard to represent. In fact, the lack of accuracy shown by classical models in modern HPC platforms has caused performance models to be partially left behind in favor of extensive experimental tests aiming to optimize the final application. This work method, however, is quite expensive in computational time; it is system dependent and does not ensure generalizable results (Hunold and Carpen-Amarie 2016). Consequently, new efforts keep emerging. The more recent model log_nP (Cameron et al. 2007) suggests a new concept, the *transfer*, a data movement between either local or remote memory entities. Sending a message is now perceived as a sequence of transfers. The cost of a transfer is not directly related to hardware but rather to the *middleware*, that is, the software that provides data movement services to the applications. The recent τ–Lop (Rico-Gallego et al. 2016; Rico-Gallego and Díaz-Martín 2015; Rico-Gallego et al. 2017) addresses some weaknesses of log_nP, including contention for the channel collectives and heterogeneous platforms.

A communication performance model represents analytically the cost of a cluster communication based on a limited set of platform-specific parameters and many models have appeared over time. The *Hockney* model (Hockney 1994) is a good starting point to get a feel for what communication performance modeling means. It is a linear model, where sending of a message has a cost $T(m) = \alpha + m\,\beta$. Here, m accounts for the size of the message, α for the network latency, and β is the reciprocal of the network bandwidth. The point here is that an adequate election of these parameters is essential to achieve a meaningful representation of the communication. The formal parameters of a model take different values in each platform; the survey discusses the empirical procedure to measure them. It is important to emphasize that this issue is critical: any change or perturbation in the measurement methodology leads to a model with a different behavior in practice and, hence, with a different prediction accuracy.

As could be expected, communication performance models play a role in the message-passing interface (MPI). Despite the appearance of alternative programming models to tackle the scalability problems posed by current clusters, the MPI (MPI Forum 2012) is, and probably will continue to be, the *de facto* communication interface used by HPC applications (Dongarra et al. 2011). Collective operations are a determining factor in the global performance and scalability of HPC software. Their underlying algorithms are designed to minimize communications, optimize the channels, or, less frequently, fit a particular network technology. The point here is that tuning the underlying

Table 1. Communication Performance Models in Chronological Order

Name	Reference	Type	Target Platform	Contribution
Postal	(Bar-Noy and Kipnis 1992)	F/Hw/G	Monoprocessor clusters	Latency-based modeling
Hockney	(Hockney 1994)	F/Hw/G	Monoprocessor clusters	Network bandwidth
LogP	(Culler et al. 1993)	F/Hw/G	Monoprocessor clusters	Basis for analysis/design of parallel algorithms in clusters
LogGP	(Alexandrov et al. 1995)	D/Hw/G	Monoprocessor clusters	Incorporates long messages to LogP
LoPC	(Frank et al. 1997)	D/Hw/G	Monoprocessor clusters	Contention on reception
Banikazemi	(Banikazemi et al. 1999)	D/Hw/G	Heterogeneous nodes in homogeneous network	Message initiation cost, constant and variable costs
Tam & Wang	(Tam and Wang 1999)	D/Hw/G	Monoprocessor clusters	Contention in network and in reception
Bhat	(Bhat et al. 1999)	D/Hw/G	Heterogeneous clusters	Collective optimization on heterogeneous networks
PLogP	(Kielmann et al. 2000)	D/Hw/G	Monoprocessor clusters	Parameters g and o dependent on message size m
LogGPS	(Ino et al. 2001)	D/Hw/G	Monoprocessor clusters	Synchronization overhead
LoGPC	(Moritz and Frank 2001)	D/Hw/S	2D Mesh networks	Contention delay per message
Kim & Lee	(Kim and Lee 2001)	D/Hw/S	Myrinet clusters	Non-linear contention cost
HiHCoHP	(Cappello et al. 2001)	D/Hw/G	Multi-level heterogeneous cluster grids	Different communication channels, segments
HLogGP	(Bosque and Pastor 2006)	D/Hw/G	Heterogeneous clusters	Per process/link parameters
Steffenel	(Steffenel 2006)	D/Hw/G	Monoprocessor clusters	Constant and variable contention factors
LogfP	(Hoefler et al. 2006)	D/Hw/S	Infiniband network	Short messages, RDMA modeling
LMO	(Lastovetsky et al. 2006)	D/Hw/G	Heterogeneous clusters	Per process/link params, fixed and variable costs
$\log_n P$	(Cameron et al. 2007)	F/Sw/G	Homogeneous clusters	Middleware costs
PLP	(Nasri et al. 2008)	D/Hw/G	Heterogeneous clusters	Segments, per process/link params
LogGPO	(Chen et al. 2009)	D/Hw/G	Monoprocessor clusters	Non-blocking comm
LogGOPS	(Hoefler et al. 2009b)	D/Hw/G	Monoprocessor clusters	Synchronization, linear overhead
LogGPH	(Yuan et al. 2010)	D/Hw/G	Hierarchical clusters	Different communication channels
$m\log_n P$	(Tu et al. 2012)	D/Sw/G	Hierarchical clusters	Middleware costs of different communication channels
Chan	(Chan et al. 2015)	D/Hw/S	Multi-level heterogeneous cluster grids	Empirically calculated point-to-point cost
τ-Lop	(Rico-Gallego et al. 2016)	D/Sw/G	Heterogeneous clusters	Middleware costs, contention, synchronization, non-p2p colls, segments

algorithms of MPI collectives is a must to quickly know their performance without painful and expensive experimental testing. In fact, currently, a key branch of interest of performance models is their ability to model collectives.

For the sake of seeing a model in action, assessing it, and making simple but significant comparisons with other models, we discuss the analytic representations of point-to-point and collective operations of the most relevant performance models. Only a small number of references can be consulted as partial or limited surveys in the field. Maggs et al. (1995) is about ancient *PRAM* (JáJá 1992) and *BSP* (Valiant 1990) computational models, and includes some network models such as *HMM* (Aggarwal et al. 1987) and *LogP* (Culler et al. 1993). The more recent article on the subject (Pješivac-Grbović et al. 2007) provides an exhaustive modeling of multiple algorithms used in MPI collective implementations using a set of models. In contrast, our survey attempts a comprehensive overview of the main works in which the models have been used to evaluate and improve the performance of a broad range of algorithms and applications. At all events, to manage the vast amount of available literature about the subject, the models discussed, considered as the most representative, are enumerated in Table 1. It provides the following:

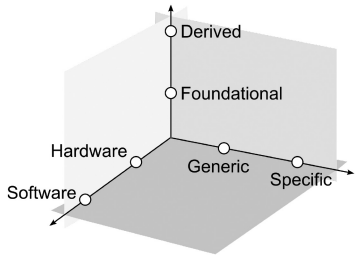

Fig. 1. A communication performance models can be characterized by the following three factors: 1) Originality, 2) Platform independence, and 3) Parameterization. These factors are orthogonal to each other.

- *Name* of the model, often an acronym composed of the initial letters of the name of its parameters.
- *Reference* of the paper where the model is introduced and described.
- *Type* of the model. Figure 1 is an attempt to classify and categorize a model in a tri-axial plot. A model can be either foundational (F) or derived (D), either hardware (Hw) or software (Sw), and either generic (G) or specific (S). We consider a model as foundational when it introduces a paradigm shift and promotes derivative models. The increasing communication costs attributed to middleware have been ignored by hardware-parameterized models, a source of inaccuracy that led to the software-parameterized models. Generic models can be adapted to different platforms. Specific models cover particular platforms.
- *Target platform*, which span mono-processor homogeneous, hierarchical, and heterogeneous clusters. By *hierarchical*, we mean clusters of regular multi-core processors, with at least two different communication channels: shared memory and network. *Heterogeneous* means clusters with different types of processors, accelerators, and communication channels.
- *Contribution* of the model. Shows the main contribution to the field. To date, the tendency has been that models add new features to previously existing models. These add-ons include the capability of representing synchronization, segmentation of messages, channel contention, specific network technology, topology, non-point-to-point-based collectives, and more.

Overall, the goal of this survey is to provide researchers with a guide to the literature, showing the scope, behavior, benefits, limitations and perspectives of communication performance models. Section 2 presents the principles of the discipline, discussing the foundational models and their original intended applicability. Section 3 explains how the models reported so far have addressed the challenges posed by modern platforms. Section 4 discusses the issues involved in the design of the parameters of a model as well as the procedure to estimate their value in each particular platform. Section 5 presents the main factors influencing the performance of a model. We offer our conclusions on the evolution of the models and thoughts on areas of future developments in Section 6.

2 FOUNDATIONAL MODELS

The evolution of the models along time makes it possible to classify them into foundational models and their extensions or derived models, which deal with issues not covered by the first ones.

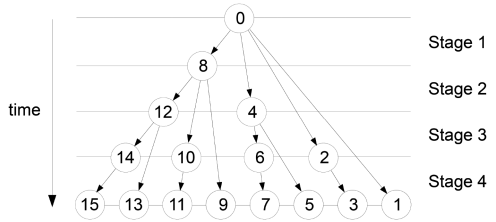

Fig. 2. Operation of a *binomial tree* broadcasting algorithm along $\lceil \log_2(P) \rceil = 4$ steps between $P = 16$ processes. A directional arrow connecting two processes (nodes in the tree) is a *point-to-point message transmission*.

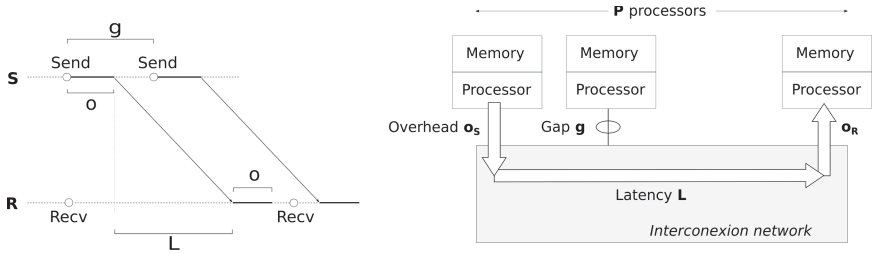

Fig. 3. The *LogP* model and its parameters.

2.1 Concepts and Formalisms

It is now more than two decades since the Bar-Noy and Kipnis *Postal Model* (Bar-Noy and Kipnis 1992) attempted to analytically model the communications of a fully connected network by introducing the issue of the *latency* of the network for first time. $T_{p2p}(m) = \lambda$. The model is asynchronous and assumes that at most one message can be sent or received by a processor per timestep. Two years later, the *Hockney* model, presented in the introduction, added bandwidth to the postal model, which limits the amount of data to be transferred at a time. The cost of a point-to-point transmission is $T_{ptp}(m) = \alpha + m\,\beta$. Because of its simplicity, the *Hockney* model has played an important role in the evaluation and optimization of any collective algorithm—in particular, those of MPI implementations. All the works apply *Hockney* assuming that a node is allowed to execute a sending operation and a receiving operation at the same time (for instance, the time for two processors to send each other a message of length m is also $T(m)$) but is not allowed to execute more than one sending operation or more than one receiving operation at the same time.

Beyond the single point-to-point transmission, to appreciate the significance of performance modeling, let us consider a more complex communication, the well-known *binomial tree* broadcasting. It proceeds according to the following algorithm. A process named *root* sends to the rest. Figure 2 shows the algorithm deployment. The process root sends a message to process $root + P/(2n)$, where P is the total number of processes and n is the current communication stage. Acting as roots of their own $P/(2n)$ node trees, these processes continue repeatedly until the leaf node is reached. The theoretical cost of a collective algorithm has been typically evaluated as the aggregate of the *Hockney* costs of its individual transmissions. Given that the height of the full tree is $\lceil \log_2 P \rceil$, everything completes in that number of stages; thus, the algorithm cost is $T(m) = (\alpha + m\,\beta) \times \lceil \log_2 P \rceil$. This result can be analytically compared with those obtained by alternative broadcast algorithms.

The convergence in the 1990s to the cluster architecture raised the development of the *LogP* model (Culler et al. 1993, 1996b). Its name is an acronym from its four parameters (see Figure 3),

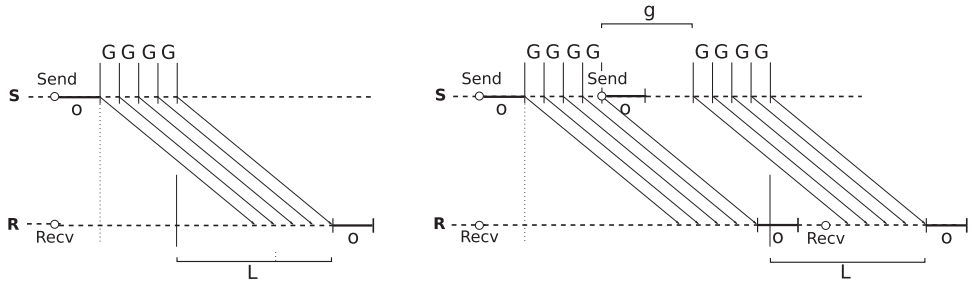

Fig. 4. Representation of the cost of a point-to-point message transmission under the *LogGP* model, one message of size $m = 5$ at the left and two messages at the right. Note that the gap g delays the emission of the second message.

L, o, g, and P, standing for *network delay*, *overhead*, *gap per message*, and the number of processors in the cluster. The overhead o is the time or cycles that the processor invests to send the message, that is, to prepare the message, enqueue it in the send queue, and signal the network interface card (NIC). It is the same for reception. Latency or network delay L is the time elapsed between the instant in which the sender processor ends the sending action and the instant in which the NIC interrupt notifies the target processor of message arrival. The actual latency that a message experiences is undefined, but L bounds above this figure for an unloaded network. The transmission of a short message has a cost of $T_{p2p} = o + L + o$. The model can be extended for further accuracy by distinguishing between the sender (o_S) and the receiver (o_R) overhead (Culler et al. 1996b). Parameter g, for gap, is the minimum time interval needed by the network card between two consecutive packet injections to the cable (likewise for receptions). Its reciprocal, therefore, gives the effective bandwidth in terms of messages per unit time. The L and g parameters configure a finite capacity network, meaning that a processor can send up to $\lceil L/g \rceil$ messages before being blocked in the network. Indeed, the key advance of *LogP* over the Postal and *Hockney* models is that *LogP* recognizes the contribution of the processor to the communication latency. The difference between o and g is the amount of CPU cycles available for the processor to do other useful computation. This is how the model exposes the overlapping of communication and computation. According to Karp et al. (1993), the Postal model "turns out to be a special case of the *LogP* model (with $g = 1$ and $o = 0$)."

LogP is important, because it became the foundation of numerous subsequent models that extend it by adding parameters for representing specific characteristics of the communication and platform. However, an important limitation of *LogP* is that it considers only short messages. The model supposes that a long message is sent as a sequence of shorter messages, using the minimum time interval g. Soon emerged *LogGP* (Alexandrov et al. 1995) to introduce a fifth parameter, the *gap per byte* G, which represents the time elapsed between the injection in the network of two consecutive bytes. The reciprocal $1/G$ is hence the network bandwidth for long messages, which corresponds to the β parameter of the *Hockney* model. Of course, G models the network bandwidth much more accurately than adding up the costs of multiple small messages limited by g in *LogP*. As an example, the left side of Figure 4 plots the cost of a point-to-point transmission as $T_{p2p}(m) = 2\,o + L + (m-1)\,G$. The right side of the figure shows the cost of two consecutive transmissions, $T_{p2p}(m) = o + (m-1)G + g + (m-1)G + L + o = 2o + 2(m-1)G + g + L$.

2.2 Application Fields

Adopting the principal computing structure in HPC, most of the literature on the application of communication performance models in clusters focuses on improving the performance of MPI

collectives. We describe noteworthy works in this area and also on optimizing the schedule of algorithms in clusters.

2.2.1 Advice on Performance of MPI Collectives. Thakur et al. (2005) use the *Hockney* model to improve the overall performance of collectives in clusters with switched networks. More to the point, the *Hockney* predictions guide the MPICH (MPICH 2013) library at runtime when selecting one among the multiple algorithms available to a collective operation depending on the message size. This work applies the *Hockney* model, making ideal assumptions that processes can send and receive messages simultaneously or there is no contention on the network. Likewise, Chan et al. (2007) use the Hockney model to analyze some MPI collective algorithms for short and long messages, proposing a parameterized family of algorithms to automate the optimization of collective communication libraries. A Hockney-based optimal broadcast is developed by Träff and Ripke (2005) and evaluated in clusters of SMP nodes. Rabenseifner and Träff (2004) discuss under the Hockney model the MPI *reduction* operations, considering fully homogeneous platforms. They propose algorithms for the specific case that the number of processes is not a power of two. The *Hockney* cost of *All-to-all* collective is discussed by Kale et al. (2003) for 2D-mesh, 2D-Grid, and Hypercube topologies in homogeneous clusters. Hatta and Shibusawa (2000) addressed the *gather* collective in heterogeneous clusters. Here, latency and bandwidth are specific to each pair of processes so that $T_{i,j}(m) = \alpha_{ij} + m\beta_{ij}$. Ooshita et al. (2002) develop a method for finding optimal broadcast trees in heterogeneous clusters. Only small messages are considered so that only the latency of the *Hockney* model is used, termed t_i (initiation cost of process i), which is calculated as half a round trip of a 4B message between two identical nodes. Hasanov et al. used the *Hockney* model for the design and analysis of hierarchical modifications of classic collective algorithms for homogeneous large-scale platforms in Hasanov and Lastovetsky (2017) and Hasanov et al. (2015a, 2015b). SimGrid is the most popular simulation tool of large-scale distributed systems. In addition to Grids, it covers Clouds, P2P, and HPC systems (Casanova et al. 2014). Its network model is the *Hockney* model. Last, but not least, the High Performance Linpack (HPL) algorithm benchmarks the Top500 list. Its scalability analysis is based on a *Hockney* communication performance model (Chou et al. 2007; Petitet et al. 2016).

Pješivac-Grbović et al. (2007) provide a thorough report of MPI collective algorithms, analyzed with *LogGP*, *Hockney*, and *PLogP*. Martin et al. (1997) use *LogGP* to study the impact of network features on the performance of applications. They conclude that applications are quite sensitive to slight alterations of overhead o and per-message bandwidth L/g but are unexpectedly tolerant to a lower per-byte bandwidth $1/G$ and to a greater latency L.

2.2.2 Algorithm Design for Optimal Communication Schedule. The idea is reorganizing the computation of the algorithm so that, as far as possible, it overlaps the communication, hence minimizing the delays produced by the latency and bandwidth of the network. The method is to express the algorithm in terms of the parameters of the model, empowering the designer to address critical performance points, avoiding the myriad of idiosyncratic machine details. Needless to say, this approach lays the foundation for devising efficient and portable parallel algorithms in a cluster. The postal model is used in Bar-Noy and Kipnis (1992) to study different broadcast algorithms and in Bernaschi and Iannello (1998) to advise on the schedule of broadcast and reduction algorithms.

The *LogP* broadcast modeling deserves to be highlighted here since it quickly provides insight on the role of the model in optimizing an algorithm in a current cluster. Look at Figure 5. Once the root process sends the message at time 0, it enters the network at time o and arrives to the target network controller at $o + L$. The corresponding interrupt copies the message to the receiver, which has it available at $L + 2o$. Note that subsequent messages depart from the root process at times $o + g$, $o + 2g$, and so on. Now, consider a type of broadcast where each receiver behaves as

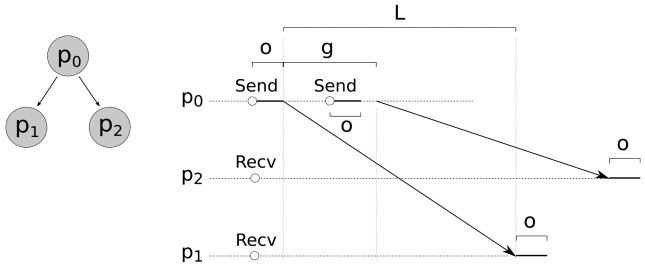

Fig. 5. The *LogP* model at work, modeling a simple algorithm, a linear broadcast between three processes (here, $g > o$). The cost of the operation is given as the time length of the longest path, $T_{Bcast} = g + 2o + L$. In general, a linear broadcast among P processes has a cost $T_{Bcast} = (P - 2)g + 2o + L$. Note that L and o determine the time to perform an individual point-to-point message transmission, while parameter g is key to making a faithful representation of a sequence of messages originating from the same source process (broadcast or scatter) or directed to the same destination process (gather). It often happens that $o > g$. In this case, after this long o experimented by a message m_2, the previous message m_1 has already been injected in the cable; thus, m_2 never experiments the gap delay. As a result, the cost analysis can ignore the gap.

the root of its own tree. Interestingly, achieving the optimal performance from this algorithm, that is, the minimum completion time, results in an unbalanced tree, because a node n_i can begin to broadcast long before another node n_j had received anything. After discussing the idea, Culler et al. (1993) present and analyze with *LogP* some example algorithms commonly used in applications, such as broadcast, summation, or FFT. Likewise, Karp et al. (1993) propose optimal scheduling under *LogP* for six different problems. Dusseau et al. (1996) use *LogP* to model a set of sorting algorithms. One of the conclusions is that *LogP*, even though it offers accurate predictions, underestimates the cost of the algorithms because it does not consider contention costs beyond its network capacity constraint. Martin et al. (1997) use *LogGP* to study the impact of network features on the performance of applications. They conclude that applications are quite sensitive to slight alterations of overhead o and per-message bandwidth L/g but are unexpectedly tolerant to a lower per-byte bandwidth $1/G$ and to a greater latency L. Santos (1999) studied the problem of broadcasting k-items from one processor to the remaining processors and proposed a slight variation of *LogP* allowing buffering of messages, under which an optimal k-item broadcast is developed. Park et al. (1996) propose the design of optimal trees based on *LogP* for the multicast communication pattern used in broadcast and scatter MPI operations. Iannello et al. (1998) show that *LogP* is a useful analysis tool for designers to get insights into the issues affecting the total communication time, using a platform with a Myrinet network. Finally, Hoefler et al. (2005) apply *LogP* to the modeling and evaluation of different algorithms implementing the *MPI_Barrier* collective.

3 KEEPING THE PACE OF THE HPC PLATFORM DEVELOPMENT

As with the case with *LogGP*, the HPC research community soon identified limitations, weaknesses, and open issues in the foundational models, and readily addressed them through derived models. It has been two decades since Tam and Wang (1999) provided insightful discussion on the similarities and drawbacks of the pioneering communication models. The cluster evolution has brought new modeling issues that could hardly be considered in 1993. We can highlight areas such as accurate MPI cost prediction, the contention for the channel, the irruption of multi-core nodes in the hierarchy of the network, or, more recently, the heterogeneity of the platform. This section summarizes proposals of derived models, with a focus on their specific abilities.

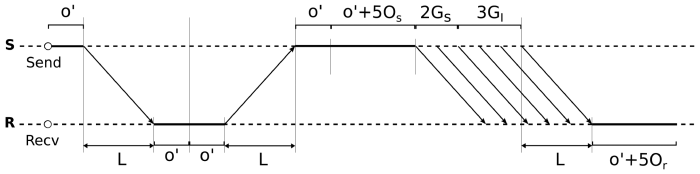

Fig. 6. Transmission of a point-to-point message of size $m = 5$ modeled in *LogGPS* under parameters ($s = 2$, $S = 4$). As $m > S$, rendezvous is used. As $m > s$, the first s bytes are sent under G_s and the rest under G_l.

3.1 Accurate Modeling of MPI Primitives

LogP/LogGP has exhibited an overall good performance for communication primitives built either at machine level or on Elan or Active Messages, both low-level communication libraries. However, it becomes inaccurate for higher-level primitives, such as those from MPI. As an example, buffer space constraints at the receiver impose the support of communication protocols in the implementations of *MPI_send*. Two of these protocols are *eager* and *rendezvous*, applied to short and large messages, respectively. In the eager case, the message is sent assuming that the target has enough store. Under rendezvous, however, the sending of a small control message precedes the delivering of the actual message. The purpose is to urge the receiver to provide enough buffering. The sender blocks until the confirmation arrives. Rendezvous incurs hence a significant added cost at the beginning of the communication, which must be considered in a model, something that *LogP/LogGP* does not do. This overload is called *synchronization* cost by Ino et al. (2001). To capture this cost, they developed the *LogGPS* model, which presents three differences with respect to *LogGP*. First, and most important, it adds the message length threshold parameter S. When $m > S$, the sender waits for an acknowledgment. Otherwise, just an asynchronous message is shipped. Second, for better accuracy, the overhead in sender is size dependent: $o_s(m) = o' + mO_s$, where o' is a constant and O_s is a factor per byte. Likewise, in the receiver, $o_r(m) = o' + mO_r$. Third, the model adds one more parameter, the length s. For $m \leq s$, G becomes G_s and G_l for $m > s$. Let us assume a system with $s = 2$ and $S = 4$. Suppose a message of size $m = 2$. Being $m \leq S$, it will be sent asynchronously. Its *LogGPS* cost is $T_{ptp}(m) = o' + 2O_s + 2G_s + L + o' + 2O_r$. If $m = 3$, as still $m \leq S$, the message will also be sent asynchronously. However, now, $m > s$, with the consequence that the third byte is sent under G_l, with cost $T_{ptp}(m) = o' + 3O_s + 2G_s + G_l + L + o' + 3O_r$. Figure 6 illustrates the cost of sending a synchronous message.

Chen et al. (2009) proposed *LogGPO* after observing that the *LogP* family fails to make satisfying predictions on the communications of MPI. The potential overlap of computation and communication may have a broad incidence on the overall performance of an MPI program, which use non-blocking communication to this end. However, the fact is that current MPI implementations fail to attain a true overlapping. The authors also point out that the reason is the significant communication overhead that can be introduced by the MPI implementation, such as the design of the communication progress engine. The issue here is that *LogGP* ignores this augmented overhead, assuming a perfect overlap degree that leads to unrealistic predictions. To tackle the problem, *LogGPO* extends *LogGP* to capture the real overhead of MPI libraries and characterizes the potential overlap between computation and communication in MPI programs. *LogGPO* introduces the overlap for sender O_s and receiver O_r; both later split into a set of parameters for representing different aspects of the non-blocking communication. These aspects are MPI *wait* time O_w, the NIC initialization time O_i, the memory copy time O_c, and the time to process a control message O'. The model defines the overlap ratio of computation to communication. The computation time C is denoted in the sender as $R_{so} = \frac{C}{C+O_s}$ and in the receiver as $R_{ro} = \frac{C}{C+O_r}$. The result is a high number of parameters. To get a flavor of the model, Figure 7 illustrates two

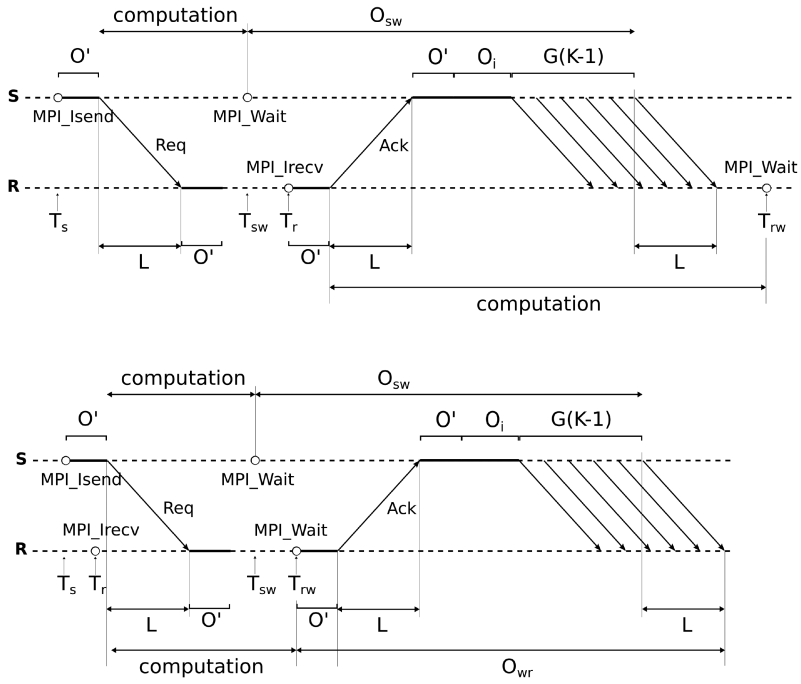

Fig. 7. A rendezvous communication of $K = 5$ bytes modeled in *LogGPO*. The plots differ on the relative arrivals of *MPI_Recv* and *MPI_Wait* in the receiver.

modeling cases of sending a message of size $K = 5$ under the rendezvous protocol with MPI asynchronous primitives. The instant of invocation of MPI_Wait makes the difference. Let us examine first the upper case. In the sender side, we have that the overhead introduced by MPI_Wait is $O_{sw} = 2O' + O_i + L + G(K-1) + (T_r - T_{sw})$; hence, $O_s = O' + O_{sw}$. In the receiver side, MPI_Wait does not introduce overhead; thus, O_{rw} is zero and, hence, $O_r = 2O' + O_{rw} = 2O'$. The modeled cost is $T_{ptp} = 2(2O' + L) + (T_r - T_{sw}) + 2O_i + G(K-1) + L$. Regarding the lower case, note that the sender side has the same behavior as before. However, in the receiver side, MPI_Wait now introduces a significant overhead $O_{sw} = 2O' + O_i + L + G(K-1)$; hence, $O_r = O' + O_{rw}$. The modeled cost is $T_{ptp} = 2(2O' + L) + O_i + G(K-1) + L + (T_{rw} - T_{sw})$.

3.2 The Network Hierarchy

Many HPC infrastructures, such as Grid'5000, couple multiple clusters via wide-area networks. A major problem in programming parallel applications for such platforms is their hierarchical network structure: latency and bandwidth of WANs often can be orders of magnitude worse than those of local networks. The *Parametrized LogP* or *PLogP* model (Kielmann et al. 2001) was conceived with the goal to optimize MPI's collectives in this scenario. An insightful experimental rationale supports the transformation of *LogP* parameters g, o_s, and o_r into piecewise linear functions of the message size m. The left side of Figure 8 shows that *PLogP* uses two differentiated parameter sets for LAN and WAN, identified by a subscript as L_l and L_w. Though the notion of latency in *PLogP* is slightly different from those of *LogP/LogGP* (see the right side of Figure 8), the gap and overhead parameters are equivalent in both models. Kielmann et al. (2000) "Like L, $g(m)$ covers all contributing factors. From $g(m)$ covering $o_s(m)$ and $o_r(m)$, follows $g(m) \geq o_s(m)$ and $g(m) \geq o_r(m)$." Under *PLogP* $T_{p2p}(m) = L + g(m)$. The right side of Figure 8 shows the parameters

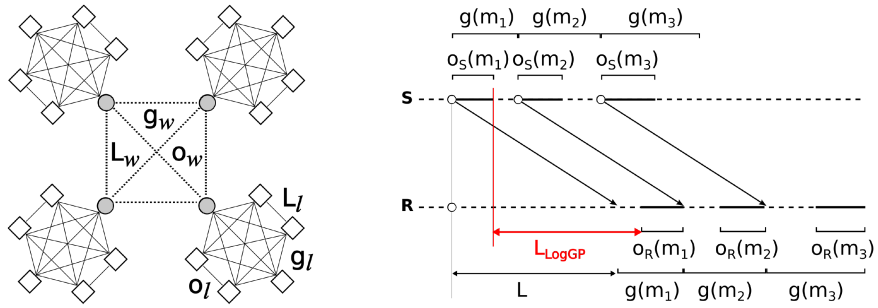

Fig. 8. On the left, the *PLogP* parameters, illustrating the difference of L between *LogP/LogGP* and *PLogP*. The right plot represents a sequence of n messages under *PLogP*. Note that the latency L contributes to the cost only at the start of the communication; hence, the cost will be $T = L + g(m_1) + g(m_2) + \cdots + g(m_n)$.

in a row of n messages of growing size. Inter-cluster Messages are split into multiple segments and sent in parallel over different WAN links. The model is applied at runtime for deciding the segment size and the optimal tree in broadcast inter-cluster operations. Both inter-cluster and intra-cluster collectives are explored by Barchet-Estefanel and Mounié (2004), with decisions based on *PLogP*.

Modern HPC clusters are composed of multi-core nodes linked by a high-performance network technology. This fact broadens the concept of "network" to a hierarchy of channels with different capabilities. Also derived from *LogGP*, Yuan et al. *LogGPH* (Yuan et al. 2010) supports the representation of hierarchical networks by recognizing the concept of "communication level." They allocate a different battery of parameters per level. Two processes running in the same socket communicate in level 1 through shared memory, in level 2 if they run in the same node but in a different socket, and in level 3 through the network. The cost of a transmission in level i of the hierarchy is $T_i(m) = 2\,o_i + (m-1)\,G_i + L_i, 0 \leq i \leq H$. *Hierarchical* algorithms are modeled in phases involving different channels. Let us consider the broadcast case. The first stage consists of an inter-node broadcast performed between selected processes, one per node. The second stage consists of a local broadcast inside each node through shared memory, where the selected node acts as the root. A *Hockney*-based hierarchical model was proposed in the mpC programming system to predict the communication cost of parallel algorithms in heterogeneous clusters of shared-memory multiprocessors (Lastovetsky 2002).

3.3 Communication Contention

Models discussed up to this point are known as *contention free*, valid as long as there is no contention on the network. That is why Moritz and Frank (2001) considers that the *LogP* family has been successful in the rather restricted boundaries of "regular applications with good communication locality and tight synchronization." Concurrent access to a shared channel, however, shrinks its available bandwidth and degrades the overall performance. Modeling this situation, for instance, is required to provide accurate cost estimation of network demanding collectives as *All-to-all*. Two types of contention can be identified (Tam and Wang 2003), *node* contention, the contention for the resources at nodes, and *link* contention, the contention for the network links.

In Distributed Shared Memory (DSM) machines, the time invested by the processor is off-loaded to a special network controller. Thus the o parameter of *LogP* is understood in Holt et al. (1995) as the mere occupancy of the network controller. The authors study the effects of o and L on the performance of a variety of communication patterns in a simulation environment. Their conclusions allow identification of situations where a controller becomes a performance bottleneck. This and some other experiences on contention effects motivated the first model that considered

contention, *LoPC* (Frank et al. 1997). The acronym misses out the letter g because it is considered negligible in the discussed platform. The send overhead is also ignored. Regarding contention, *LoPC* considers it only in the receiver node. Processes communicate via active messages. As messages are short, the contention is only attributable to the battle for the processor between the threads that handle the incoming messages. The cost of a transmission can be represented as $T_{ptp} = S_l + S_o$, with S_l the latency, with the same definition as L in *LogP* and with S_o the overhead in the reception processor, including the contention.

LoGPC (Moritz and Frank 2001) extends *LogGP* to mesh networks to represent the impact of network contention C_n; thus, $T(m) = o_S + L + (m-1)\,G + C_n + o_R$. Here, n is a dimension of a K-ary N-cube mesh (N dimensions and K nodes per dimension). C_n comes from the average switch delay, derived solving the linear system resulting from a $M/G/1$ queuing model. Unfortunately, the number of parameters raises significantly. The authors observed that network contention represented up to 50% of the overall execution time in three numerical codes in the Alewife machine.

Tam and Wang (1999) define a model, rooted on *LogP*, that also exposes the contention explicitly. The model splits the overhead in the receiver into two parameters—O_r and U_r. O_r—representing the cost of the asynchronous receiving event in the kernel, while U_r is the added cost in the user space. A single global switch models the network as a complete graph. The latency parameter $L(m,p)$ includes the network contention by its dependency on the number of processes p. L is influenced by the B_L parameter, which represents the available buffers in the central switch, which cannot serve two consecutive packets within the gap period g. The cost of a transmission is modeled as $T_{ptp}(m,p) = O_s(m) + (k-1)g(m) + L(m,p) + O_r(m) + U_r(m)$, where $k = m/b$ is the number of fragments of the message, with b being the size of the fragment. This cost is considered to be that of a *data movement* through the network between two remote processes. The model also considers parameters for intra-node communications. Thus, though being a hardware model, the data movement abstraction makes it a precursor of the middleware performance models.

Next, we briefly review different interesting works on modeling contention in high-performance networks. The traditional linear point-to-point cost model used in a Myrinet cluster (with wormhole routing) is $T_{ptp}(m,h) = t_s + h\delta + mf$, where t_s is the send and receiving overhead of *LogP*, f the reciprocal bandwidth, h is the length of the path through the switches of the network, and δ the delay in the switch. Kim and Lee (2001) argue that this model is insufficient in the presence of synchronization protocols, the varied behavior of the switches and the contention in the links. Thus, they derive a piece-wise linear model by adjusting measured delays. Suppose that m is in the interval $[x_0, x_1]$, which has a slope S. In the absence of contention, $T_{ptp}(m,h) = T_{ptp}(x_0,h) + S \times (m - x_0) + \delta(h-2)$. Under contention, $T^C_{ptp}(m,h) = \eta_{max} T_{ptp}(m,h)$, where η_{max} is the upper bound of messages contending for using any of the links on the path. They use BIP, a low-overhead user-level communication library for Myrinet. The model is tested against an "ad hoc" and a binomial broadcast, with suggestive rather than conclusive results.

Though perceived as a first approach, Martinasso and Méhaut (2005) proposes an interesting contention model in a single-switch Myrinet network of 104 bi-Itanium2 two-core nodes, this time using MPI as a user-level library. They capture the impact of both node and link contention using just two *hockney* models, one contention-free and another on node contention. The authors extend the study to Infiniband networks in Martinasso and Méhaut (2011). Jerome et al. (2008) perform a quite similar study on bandwidth sharing for multiprocessor clusters connected by Ethernet, Infiniband, and Myrinet networks. A study of the mechanism of flow control of each type of network gives place to its particular model. Experiments of resource sharing categorize various conflict types and output "penalty coefficients."

Steffenel (2006) and Barchet-Steffenel and Mounié (2006) dwell on the incompetence of traditional performance models to predict the cost of the *All-to-all* collective. They try to solve this problem with the identification of the *contention network signature* parameters γ and δ of a given network. γ represents the ratio of the theoretical lower bound of the communication operation and the actual measured execution time, and δ is an extra supplementary contention factor of observed slopes. Both parameters are functions of the message size and the number of processes. They are used to augment a linear model previously determined, with the goal of fitting the real cost of *All-to-all*. *PLogP* or other models are used to estimate the contention-free communication cost. The network contention is then linearly incremented. The cost of the transmission is expressed as $T_{ptp}(m) = L + g(m) + \gamma + \delta$ under *PLogP*.

Bidirectional TCP traffic introduces contention over Ethernet channels. Zhu et al. (2013) have developed a communication model that characterizes the asymmetry of this contention.

3.4 Platform Heterogeneity

LogP originally aimed at clusters of homogeneous single-processor nodes linked by a single network. Banikazemi et al. (1998), however, targeted their work to clusters of heterogeneous workstations. Their work reveals how the overhead of MPI point-to-point transmissions can vary significantly depending on the capabilities of the nodes. In other words, each processor i has a specific t^i_{ini} *message initiation cost*. Since collectives involve more than one workstation, the question arises as to whether the heterogeneity can be used to the advantage of implementing them faster. Two new approaches are proposed to implement broadcast and multi-cast with reduced latency. Later, Banikazemi et al. (1999) propose a model with the goal of optimizing collectives in the cited platforms. It is based on the definition of the cost of a point-to-point transmission being $T_{ptp}(m) = O_s(m) + O_t(m) + O_r(m)$, where the three O parameters are linear functions of the message size m. Thus, the overhead in the sender is $O_s(m) = S_c + S_m\, m$, the overhead in the receiver $O_r(m) = R_c + R_m\, m$, and the overhead of the transmission $O_t(m) = X_c + X_m\, m$.

Computational grids emerged almost two decades ago as an infrastructure that connected distributed computational sites worldwide. This system is heterogeneous both in nodes and networks. Bhat et al. (1999) extend the *Hockney* model with communication costs summarized by inter-node cost matrices. The cost of a transmission between process i and process j is defined as $T_{ij} + \beta_{ij} \times m$, with T_{ij} representing the start-up time attributable to the i process, including the latency of the link connecting the processes i and j. They provide several heuristics to find efficient broadcast and multi-cast over grids with heterogeneous networks. Bhat et al. (2003) also studied the all-to-all collective in these systems.

HiHCoHP (Hierarchical Hyper-Clusters of Heterogeneous Processors) is a model proposed by Cappello et al. (2001, 2005) aimed at modeling multi-level hierarchical networks of heterogeneous clusters. A transmission between sender P_s and receiver P_r traverses multiple levels in the network hierarchy, ordered from the lower-latency/higherbandwidth level. The communication costs are represented by the message processing costs at the processors in level k of the hierarchy $\sigma_s^{(k)}$ and $\sigma_r^{(k)}$, the costs of packaging and unpackaging each packet of the message $\pi_s^{(k)}$ and $\pi_r^{(k)}$, and three parameters representing the delays of the network: (1) the network latency $\lambda^{(k)}$ that includes the latencies of the levels lower than k; (2) the network end-to-end bandwidth $\beta^{(k)}$, including the lower-level bandwidths; and (3) the network capacity $\kappa^{(k)}$ as the maximum number of packets in transit at once through the level k of the network. The cost of sending a message in p packets in a pipelined network can be represented as $T(m) = \sigma_s^{(k)} + \pi_s^{(k)}\, p + \lambda^{(k)} + \frac{p-1}{\beta^{(k)}} + \sigma_r^{(k)} + \pi_r^{(k)}\, p$, given a per-packet delay of $1/\beta^{(k)}$. The model also includes the processor time to perform a unit of work as ρ_i for each P_i. *HiHCoHP* is applied to broadcast and reduction operations.

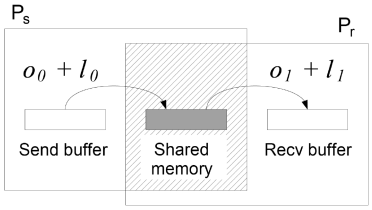

Fig. 9. Concept of *message transmission* in log_2P. P_s is the sender process while P_r is the receiver. Sending a message through shared memory imposes two *transfers* in the channel. Physical memory reads and writes have slightly different performance, thus, the overhead and the latency parameters are different. Notwithstanding, considering them the same is a reasonable simplification here.

HLogGP (Bosque and Pastor 2006; Bosque and Perez 2004) is an offspring of *LogGP* that addresses the heterogeneity of both processors and channels. *HLogGP* extends the o_s, o_r, and g scalar parameters of *LogGP* as vectors of p components, p being the number of processors. On the other hand, L and G are now matrices of $p \times p$ components (one per pair of processors). The model includes the processor-dependent computational power as a new parameter denoted by P_i. The cost of a transmission between processors i and j is $T_{i\to j}(m) = L_{ij} + o_{S_i} + o_{R_i} + (m-1)\,G_{ij}$. The authors provide a methodology for estimating the value of the parameter for each pair of processors in a small cluster, and they are applied to the prediction of the execution time of a matrix multiplication code and to a volumetric magnetic resonance image compression application.

Lastovetsky et al. (2006) address the cost of communications on a switched Ethernet network connecting a set of heterogeneous processors. The model proposed, *LMO*, discusses the impact of this heterogeneity on the communication costs of a battery of operations, both of point-to-point and collective. Just like *LogGP*, *LMO* conscientiously segregates processor and network costs. The cost of the transmission between the processors i and j is $T_{i\to j}(m) = C_i + m\,t_i + C_j + m\,t_j + L_{ij} + m/\beta_{ij}$. Parameters C_i and C_j are specific delays attributable to each particular processor, reflecting the processor heterogeneity. t_i and t_j are specific per-byte costs, which reflect channel heterogeneity. The *fixed network delay* parameter L_{ij} introduces additional flexibility in expressing the cost of collective operations. It is added to the original model by Lastovetsky et al. (2009). β_{ij} is the transmission rate of the channel. *LMO* and its parameter measurement procedure are discussed in detail by Lastovetsky and Rychkov (2007) and Lastovetsky and Rychkov (2009).

Nasri et al. (2008) propose a model called *PLP* for predicting the communication cost in hierarchical and heterogeneous mono-processor platforms. The model takes into account the multiple levels of a hierarchical structure of a network and models the point-to-point cost as $T(m) = L + P_s + P_r$. P_s and P_r are the times spent in preparing the transfer at sender and receiver processors, and are split up in multiple costs represented by processor-dependent parameters. Network latency L is defined as $L = (n-1)\,G + L_t$, with G being the inter-frame gap, that is, the gap cost in sending the n consecutive Ethernet frames into which a message is divided, and L_t the latency cost of a frame transfer. The *PLP* model is applied to point-to-point and collectives.

3.5 The Middleware Costs

The general adoption of clusters of multi-core machines has seen a rise in new approaches to performance modeling. *Memory logP* (Cameron and Sun 2003) augments *LogP* to estimate the cost of a communication inside a memory hierarchy. Cameron and Ge (2004) and Cameron et al. (2007) propose log_nP, a model addressing the *middleware* costs of the communication (see Figure 9). Derived from the previous work on *memory logP*, its main idea is that in a point-to-point transmission, the message progresses as a succession of *transfers* (copies) through intermediate

buffers between the end-to-end buffers. The aggregate of the costs of the n individual transfers yields the cost of the transmission: $T(m) = \sum_{i=0}^{n-1} (max\{g_i, o_i\} + l_i)$, where o (*overhead*) is the per transfer time dedicated by the CPU given that the message is contiguous. The *gap* g is o plus the resource contention. Often, a message is stored in non-contiguous or strided ways. In these cases, an additional packing and unpacking burden appears that is shouldered by the CPU. This overload is modeled by the *latency* l parameter. Under the simplifications $max\{o_i, g_i\} = o_i$ and $l_i = 0 \; \forall \, i$ for contiguous messages, the model produces a transmission cost in shared memory given by $T(m) = 2 \times o = o_{mw}$. In a network, the point-to-point communication can be represented as the sequence of three transfers, with cost $T(m) = o'_{mw} + o'_{net}$, with o'_{mw} encompassing the cost of a local transfer to/from the NIC in both nodes and o'_{net} the cost of the network transfer. Although simple, the *transfer* concept conveniently suits the MPI blocking communication representation when no segmentation is present. Notwithstanding, note that these foundations lead to oversimplified cost expressions of collectives and more general algorithms. For instance, the cost of the binomial tree broadcast with contiguous messages in shared memory is modeled, similarly to *LogGP*, as $T_{bcast}(m) = \lceil \log_2(P) \rceil \times T(m) = \lceil \log_2 P \rceil \, (2 \times o)$.

Tu et al. (2012) consider that the new clusters of multi-core nodes are not addressed appropriately by the former models, with the main consequence that they are unable to predict the performance of collectives with enough accuracy. They propose the $mlog_nP$ model. It extends log_nP in order to distinguish the assortment of m communication *channels*[1] present in the platform and captures their hierarchy. The cost in channel c $(0 \leq c < m)$ is $T^c = \sum_{j=0}^{n^c-1} o^c_j$. Regarding n^c, it belongs to the vector $n = \{n^0, n^1, \ldots n^{m-1}\}$, which has one component per channel. Each component indicates the number of transfers experienced by a message before reaching the target buffer. The overhead o^c is the per-transfer processor load. Note that the channels used by the processes are determined by their locality, as are the associated costs. When only one channel is considered, log_nP and $mlog_nP$ become equivalent. Both models are applied to the basic flat tree and binomial broadcast operations.

Rico-Gallego and Díaz-Martín (2015) introduce the τ*-Lop* model. It addresses the representation of contention in the channels. The model is tested in a large shared memory system being applied to a wide spectrum of collective algorithms, either supported by point-to-point primitives or ad-hoc shared memory techniques, like those found in Open MPI. Rico-Gallego et al. (2016) extends τ*-Lop* to represent the network communication. It models and empirically studies the contention in multi-core clusters, with emphasis on their combination of channels of different natures and performance. In τ*-Lop*, a *message transmission* is a concatenation of possibly concurrent progressing *transfers*, with a cost defined as $T^c(m) = o^c(m) + \sum_{j=0}^{s-1} L^c_j(m, \tau_j)$. The *overhead* $o^c(m)$ is the time interval from the transmission start-up to the instant in which data begin to be injected in the c channel. The *overhead* not only covers the software stack cost but also that of the agreement protocol, as rendezvous. L^c, known as *Transfer time*, represents the time it takes to copy the message data between two buffers of the c channel. τ*-Lop* captures the fact that a transfer may share the c channel with other concurrent transfers. As a result, L is not just a function of m but also a function of the number τ of such competing transfers. The number of transfers s that makes up a transmission is channel dependent. As an example, the model represents the cost of a segmented transmission in shared memory $(c = 0)$ through shared buffers of size S as $T^0(m) = o^0(m) + 2\,L^0(S, 1) + (k - 1)\,L^0(S, 2)$, shown in Figure 10. τ*-Lop* introduces the concept of *concurrent transmissions* for better accuracy and expressive power when modeling collective operations. The expression $A \,\|\, T^c(m)$ means the cost of A concurrent transmissions T contending

[1]The term *channel* is used in this article instead of the original *level* used in $m\log_n P$ model.

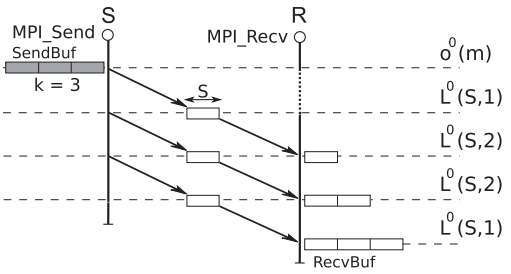

Fig. 10. τ-*Lop* cost analysis $T^0_{p2p}(m)$ of a shared memory transmission of a message split up in $k = 3$ segments of size S. The transmission entails four steps and six transfers (arrows).

for the c channel. For instance, the *Binomial Tree* algorithm plot in Figure 2 shows a cost given by $\Theta_{Bin}(m) = \sum_{i=0}^{\log_2 P-1}[2^i \parallel T^c(m)]$. As can be seen, τ-*Lop* captures that each stage doubles the number of involved processes, making the contention grow exponentially. τ-*Lop* is applied to predict the cost of a wide collection of MPI collective operations and the SUMMA (Van De Geijn and Watts 1997) kernel in Rico-Gallego et al. (2016). τ-*Lop* is extended in Rico-Gallego et al. (2017) with a set of formal assumptions to cope with complex communication cost expressions appearing in heterogeneous environments, where messages of different lengths progress simultaneously (sometimes concurrently) through different communication channels. The model is applied to predict the costs of heterogeneous SUMMA and wave equation kernels, using 1D and 2D data partitions. Results are evaluated on a cluster of CPUs and GPUs connected by both Ethernet and Infiniband networks.

3.6 Scalability

Hoefler et al. (2010) present a simulation framework, *LogGOPSim*, to evaluate parallel algorithms at large scale. Its main goal is to study the scalable behavior of collective MPI algorithms. The simulation framework departs from a Group Operation Assembly Language (GOAL) (Hoefler et al. 2009b) representation of the communications in an algorithm in order to evaluate its performance. The authors propose a new model, *LogGOPS*. It is basically *LogGPS*, with emphasis on its per-byte overhead parameter O. The cost of a point-to-point transmission under this model is represented as $2\,o + L + (m-1)\,O + (m-1)\,G$. *LogGOPS* is applied to linear (gather and scatter), binomial (broadcast) and dissemination communication patterns.

3.7 Domain Generality versus Specificity

Martinasso and Méhaut (2011) and Tam and Wang (1999) point out a trend toward two extremes. Models are either too general and theoretical or too adapted for a particular platform or application. *LoP* (Hoefler and Rehm 2005), for instance, is a more accurate model than *LogP* for small messages in Infiniband networks. Departing from an empirical study of the platform, the authors derive a model characterized by the parameter set $(\lambda_1, \ldots, \lambda_6)$. A fitting procedure estimates these six values, adjusting them to the output of a set of benchmarks executed on the platform. Due to the complexity of *LoP*, Hoefler et al. (2006) propose *LogfP* with the additional goal of gaining insights on the hardware parallelism of Infiniband that previous models ignore. *LogfP* "encourages the programmer to use the inherent hardware parallelism in the transmission of small messages." Compared to *LogP*, *LogfP* "considers that sending the first f messages is for free." The ability of Infiniband to send small messages at once (up to f, where the network saturation starts appearing) makes g zero. Furthermore, in RDMA-based Infiniband transmissions, the receiver overhead o_R is not applicable. The parameters of the model are measured using a round-trip time (RTT)

benchmark with 1-to$P$ and P-to1 transmission patterns. According to the authors, *LogfP* "made it possible to enhance the performance of the barrier operation for InfiniBand up to 40% in comparison to the best known solution." Hoefler et al. (2008) use *LogfP* to characterize the cost introduced by multi-stage switches.

Chan et al. (2015) propose an empirical and specific performance model for predicting the overall time of execution of mesh-centric codes on a network composed by heterogeneous clusters. They evaluate the model in a small-scale cluster for a finite element application. The execution time for a process i is $t_i^{exec} = (t_i^{comp} + t_i^{comm} - t_i^{saving}) \times \Delta$. t_i^{comp} is the computation time and t_i^{saving} is the time saving in communication time due to the overlap of communication and computation. Δ is the number of iterations of the algorithm. The communication time t_i^{comm} is defined as the addition of the transmission costs of a process i through the L levels of the network hierarchy to all its neighbors in each level, defined as the set $\{d_i^l\}$. The communication pattern in the evaluated application is the *exchange* pattern. Consequently, the time is multiplied by 2. Finally, we have that $t_i^{comm} = \sum_{l=1}^{L}(2 \times \sum_{j=1}^{d_i^l} t_{ij}^l)$, where $t_{ij}^l = t_{avg}(\beta \times b_{ij}^l)$ is the average time spent by a process i in sending b_{ij}^l elements of β bytes to a process j. It is empirically calculated and statistically adjusted using a benchmark that ensures the contention in the communications.

4 EXPERIMENTAL METHODS TO BUILD MODELS

Thus far, we have introduced the most relevant models and their parameters. In this section, we focus on the methodologies proposed for measuring the parameters, also known as the methods, to build the model experimentally.

When we compared the analytical description of a model and the experimental description to construct it, we found that there is sometimes a large gap between the two. Typically, the complexity of experimentally building a model is quite high. In many cases, extra intermediate parameters, which we call empirical parameters, are introduced to experimentally determine the values of one or more analytical parameters of a model. These extra empirical parameters account for the idiosyncrasies of the underlying MPI implementation or the nature of the network. Thus, one of the key qualities of a model is its tunability, which would allow its construction with low experimental complexity preferably via use of empirical parameters that can be determined accurately but at low cost. To illustrate, we consider three examples. Culler et al. (1996b) in their experimental construction of *LogP* model for Active Messages use an intermediate parameter Δ, representing a controlled amount of computation, to determine o_r. One can observe in the popular MPI implementation MPICH and Open MPI that the underlying algorithm of a collective changes depending on message size. Threshold parameters are introduced by Lastovetsky and Rychkov (2009) to characterize these sizes. Rico-Gallego et al. (2016) use two parameters, H and S, to represent the switch between *eager* and *rendezvous* protocols and message segmentation, respectively, in the experimental construction of τ*–Lop*. When presenting an experimental method to build a model, the techniques used to improve its accuracy on various behaviors of the underlying implementation should be discussed, such as buffering, protocol changes (e.g., from *eager* to *rendezvous*), transfer of long/bulk messages, and non-linearity that may arise owing to variations/fluctuations in the network.

We would like to highlight one disconcerting commonality of most measurement methods, although some of the works we surveyed do a commendable job at presenting them. While most of the presentation in research work is spent in introducing the model and theoretical analysis of algorithms using the model, the same rigor is not followed while explaining the measurement method for the model. In most cases, considerable imagination is demanded on the part of the reader to infer how the model parameters have been measured. This is because the method is

either not presented using pseudocode (as one would for an algorithm) or is not formulated in the form of simple mathematical equations, which would have enhanced its understanding by a discernible reader. We conclude, hopefully not sounding too pedagogical, by saying that a measurement method of a model that is poorly explained will only hinder the practical adoption of the model.

4.1 Some Methods Found in the Literature

In this section, we present the measurement methods for a few notable models to demonstrate how the authors of the models have taken into account some of the myriad complexities involved in the accurate and reliable measurement of their model parameters.

Culler et al. (1996b) describe an experimental methodology to build their *LogP* model for three platforms whose communication layer is implemented using Active Messages. The first step is to obtain the RTT using a single Active Message *request-reply* operation, with cost $RTT = 2 \times (o_s + L + o_r)$. Then, they execute a sequence of n requests to measure the average time per request or message cost. For small n, the message cost is $RTT = o_s$ since the sender is busy issuing requests without processing any replies. For larger n, one or more replies arrive during the issuing of requests. The processor then spends o_r to process each reply. Eventually, messages will saturate the network completely, in which case the message cost becomes g. Consequently, the o_r cannot be measured directly using RTT because the request is idle for some time waiting for a reply and then spends o_r consuming the reply from the network. Therefore, to measure o_r, the authors introduce a controlled amount of computation Δ between the messages. Then, the average message cost is plotted as a function of n for different values of Δ. From these plots, g is calculated to be the asymptotic value of the plot for $\Delta = 0$. The value of L is then obtained as $L = (RTT/2) - o_s - o_r - g$.

To build the *LogGP* model, Hoefler et al. (2009a) present an elaborate measurement method. The fundamental building block of their method is called "the parametrized round trip time (*PRTT*) of a *ping-ping* message." $PRTT(n, d, m)$ represents the *PRTT* of n transmissions of messages of size m from the sender to the receiver and the final response message to the sender. d is an artificial delay between sending messages, which avoids network saturation. The *PRTT* of a single ping-ping message with delay $d = 0$ is described as "$PRTT(1, 0, m) = 2 \times (o_s + L + o_r + (m - 1)\, G)$." Under variable delay, $PRTT(n, d, m)$ can be written as "$PRTT(n, d, m) = PRTT(1, 0, m) + (n - 1) \times \max\{o_s + d, g + (m - 1)\, G\}$," which allows calculation of the model parameters from the derived definition "$\max\{o_s + d, g + (m - 1)\, G\} = \frac{PRTT(n,d,m) - PRTT(1,0,m)}{n-1}$". If d is chosen such that $d > g + (m - 1)\, G$, then the send overhead o_s is calculated; otherwise, the values of the parameter g and G are obtained by using least-squares fit for previous linear function. The receive overhead o_r is measured using the approach by Kielmann et al. (2000). The latency parameter L is simplified as *PRTT* of a small $m = 1$ message, $PRTT(1, 0, 1)/2$.

Kielmann et al. (2000) present a very comprehensive measurement methodology to build their *PLogP* model. It is based on two processes, *measurer* and *peer*, and the average round trip time $RTT_n(m) = L + (n \times g(m)) + L + g(0)$ of sending n messages of size m by the *measurer* to the *peer*, which acknowledges them with a single empty message. Using $RTT_n(0)$, n starts as 10 and is doubled until saturation is assumed to be achieved, at which point the value of $g(0)$ is determined. Using $g(0)$, Latency L is determined applying the previous $RTT_n(0)$ definition, and the gap per message $g(m)$ for $m > 0$ by using $RTT_n(m)$. It is not clear how the send overhead $o_s(m)$ is measured. The receive overhead parameter, $o_r(m)$, is determined using one more round trip where the *measurer* sends a 0-bytes message, waits for the previously determined $RTT(m)$, and then consumes the m-bytes message that is already waiting in the receive buffer. They also provide a simple conversion table to calculate the *LogGP* parameters from the *PLogP* parameters.

Lastovetsky and Rychkov (2007) and Lastovetsky et al. (2009) describe a very comprehensive experimental methodology to build their *LMO* model on a heterogeneous cluster of sixteen nodes. In a generalized p-node cluster, there are $2p + p^2$ unknowns: p fixed-processing delay parameters C_i, p variable-processing delay parameters t_i, and p^2 bandwidths β_{ij} between each pair of processors i and j. In order to frame a sufficient number of equations, the authors use round-trip operations and design additional experiments, called point-to-two experiments, which consist of communications from one processor to two other processors and back. A point-to-two experiment is essentially a linear scatter followed by a linear gather. However, in their experimental cluster with a switched network, they observed that collective operations behave differently for different message size ranges. Hence, they introduce extra parameters that are intrinsic to their measurement method to account for this irregularity.

Cameron et al. (2007) use a set of micro-benchmarks based on an enhanced *mpptest* package to build their log_nP model experimentally. As an example, the estimated execution time of a point-to-point network transmission in their log_3P model is equal to $o_{mw} + o_{net}$, with $o_{mw} = 2\,o$. To obtain the value for o_{mw}, they use a round-trip transfer of contiguous messages from a process to itself, called the send-to-self operation, and modeled as $T_{0,0}(m) = o_{mw} + T_{mem}$. The cost of the memory copy T_{mem} is measured for different message sizes. The value of o_{mw} is calculated as $o_{mw} = T_{0,0}(m) - T_{mem}$. Then, the value of o_{net} is obtained from a round-trip transfer of contiguous messages from a process to another process as $o_{net} = T_{0,1}(m) - o_{mw}$. It should be noted that o_{mw} and o_{net} are functions of the message size.

Rico-Gallego et al. (2016) describe an experimental methodology for building their τ*–Lop* model on a pair of multi-core clusters. There are two parameters to be estimated in the model per communication channel c in the platform, the overhead, and the transfer time. The overhead time $o^c(m)$ is assumed to be contention independent. As it comprises the protocol and software stack costs, the parameter is measured depending on the protocols implemented in the MPI libraries used, which are usually *eager* and *rendezvous*. Round-trip time $RTT^c(m)$ is used to measure the shared memory overhead for all message sizes and network overhead for messages following the *eager* protocol. A simple ping operation $Ping^c(m)$ using the MPI synchronous send operation MPI_Ssend is used, which enforces the use of *rendezvous* protocol. The authors devise an operation $Ring^c_\tau(m)$ to measure the transfer time parameter $L^c(m, \tau)$ for a message size m where they ensure that τ processes transfer data concurrently. It is implemented using MPI_Sendrecv, where process P_i receives from P_{i-1} and sends to P_{i+1} at the same time.

4.2 A Framework to Assess a Building Method

There are several categories to assess the methodology adopted to measure the parameters of a model. We organize them under an acronym called "MUOPIA" (pronounced as *Myopia*). It is certainly reasonable to expect that any research work proposing a model should document an experimental method to determine the values of its parameters accurately and precisely. We also believe that if a model is to be considered comprehensive, the methodology to build it should tick all the boxes, that is, satisfy all the goodness criteria in each category. However, we do understand that the model and the methodology to build it may not address all the pertinent issues in each category in a single research work (owing to space and time constraints). Therefore, we also attempt to make aware models, which are able to do this over a course of time. We consider a measurement methodology to be *myopic* (or suffering from MUOPIA) if it fails to meet the goodness criteria in the majority of the categories. The MUOPIA categories are:

- *Method:* This describes the experimental method used to measure the parameters. The authors of the model must clearly define the meaning of each parameter for the

experimental platform selected and how to measure it. A good method should have low implementation complexity, thereby ensuring easy reproducibility. Apart from documentation of the method, a respectable practice is to make available the software that automates the method.

- *Uncertainty:* It needs no reiterating that a measurement method must be statistically sound. Hunold and Carpen-Amarie (2016) go over in excruciating detail the pitfalls that designers of MPI measurement methods must avoid as well as the statistical design of MPI experiments. Ideally, each parameter (not necessarily real valued) in the model must be represented by an interval $(\bar{a} \pm \sigma_{\bar{a}})$, where $\bar{a}$ is the estimated value and $\sigma_{\bar{a}}$ is the estimated standard error. A good method would also quantify how the error scales with system size. In this category, we also discuss the techniques used to improve the accuracy of the method.
- *Overhead/Complexity:* We will focus primarily on two complexity measures. One is the time taken to build the model. The other is the number of needed messages. A large execution time would prohibit its employment in networks with highly fluctuating characteristics where speed and adaptability of the measurement method are essential. Too many messages can flood the network; if this happens too often, its overhead can be prohibitively high and also intrusive.
- *Portability/Platform Specificity:* A measurement method, which is highly tailored or locked to a particular underlying software implementation and a platform, will be non-portable.
- *Intrusiveness:* There are two types of interferences, inherent and causal. The first type is inherent in a measurement method and needs to be recognized and corrected post-measurement. The causal interference is induced by the measurement method into the ambient communications in a real-life system. The communications during model construction are separate from the communications during the execution of application that uses the information from the model. The model construction can be static or offline. However, a self-adaptable application can use an adaptive and dynamic communication model. This property states that in a network that is not dedicated, which is the case for any cluster (most definitely for WAN), a measurement method that floods the network frequently is considered highly intrusive. A method that uses a minimal set of communications to determine the values of the parameters is considered less intrusive. In any case, the authors should ideally spend adequate efforts explaining how their model addresses this concern. A highly intrusive measurement method can cause undue interference (or add noise) to other communications in a real-life system. Even if a method does not consider intrusiveness explicitly, the consideration could be inherent in the way the model is constructed (e.g., LMO (Lastovetsky and Rychkov 2009)). For instance: using only a sufficient set of equations to determine the parameters of a model.
- *AfterMath:* This represents the aftermath of the analytical and experimental description of constructing a model. It includes the following (not a comprehensive list): (a) data cleaning and posterior statistical tasks performed to remove noise and thereby reduce the error of measurements, (b) efficacy of the model in reducing the execution times of applications employing communication operations, (c) analysis of the model predictions, (d) summary of the maximum and average prediction errors of the model, and (e) automation of the model construction via a software package. While it is too unreasonable to ask a model to fulfill all objectives in this category, we expect a model to have performed good *AfterMath* if it has reported average and maximum prediction accuracies and if it has demonstrated improvements in execution times of applications by employing communications optimized using the model.

Table 2. Salient Characteristics of the Measurement Methods to Build Communication Performance Models

Name	Method	Uncertainty	Overhead	Portability	Intrusiveness	AfterMath
Postal	✘	✘	✘	✘	✘	✘
Hockney	✔	✘	✘	✔	✘	✘
LogP	✔	✔	✘	✔	✔	✔
LogGP	✘	✘	✘	✘	✘	✘
LoPC	✘	✘	✘	✘	✘	✘
Banikazemi	✔	✘	✘	✔	✘	✔
Tam & Wang	✘	✘	✘	✘	✘	✘
Bhat	✘	✘	✘	✘	✘	✘
PLogP	✔	✔	✔	✔	✔	✔
LogGPS	✔	✘	✘	✔	✘	✘
LoGPC	✘	✘	✘	✘	✘	✘
Kim & Lee	✔	✘	✘	✔	✘	✔
HiHCoHP	✘	✘	✘	✘	✘	✘
HLogGP	✔	✘	✘	✔	✘	✔
Steffenel	✔	✘	✘	✔	✘	✔
LogfP	✘	✘	✘	✘	✘	✘
LMO	✔	✔	✔	✔	✔	✔
$\log_n P$	✔	✔	✘	✔	✘	✔
PLP	✘	✘	✘	✔	✘	✘
LogGPO	✘	✘	✘	✘	✘	✘
LogGOPS	✔	✘	✘	✔	✘	✘
LogGPH	✔	✔	✔	✔	✔	✔
$m\log_n P$	✔	✘	✘	✔	✘	✔
Chan	✔	✔	✔	✔	✔	✔
τ-Lop	✔	✘	✔	✔	✔	✔

Note: ✓ means specified and ✘ means not specified.

Table 2 contains a summary of the salient characteristics of measurement methods described in some notable research works. When a model deals with a category exclusively, we give it a ✓ for that category. If it does not consider a category in any way, we give it an ✘ for that category. However, for some models, some of the categories are inherently dealt with in the design and implementation of the method, in which case they deservedly get a ✓.

4.3 Best Practices for a Measurement Method

In this section, we describe a procedure that will showcase the best practices that should be applied for determining the values of the parameters of a communication model on a given platform.

A communication model is defined by a set of analytical parameters that represent the public interface of the model and are used to derive analytical expressions for the cost of point-to-point and collective communication operations. For example, *LogP* has a set of three primary parameters, {L,o,g}; *τ-Lop* is defined by a set of two parameters, $\{o^c(m), L^c(m,\tau)\}$. The experimental method to determine them is an iterative process. At each step, to determine the values of one or more primary parameters, the model is extended or applied typically in two ways. In one application, additional empirical parameters are introduced. These parameters are specific and internal to an experimental procedure and are sometimes designed based on the intuition of the experimenter. For example: Culler et al. (1996b) in the experimental method to build their *LogP* model introduce a secondary parameter Δ, the amount of controlled computation between messages, to measure o_r, the receive overhead. Hoefler et al. (2009a) in the measurement method

to build their *LogGP* model use a parameter d, the artificial delay between messages, to determine the send overhead o_s. The other is the use of the model to derive analytical cost expressions for collective communication operations (e.g., using the model's point-to-point parameters) to compose a consistent system of equations to solve for the parameters. For example, Lastovetsky and Rychkov (2007) and Lastovetsky et al. (2009) devise a point-to-two experiment, which is essentially a linear scatter followed by a linear gather. To summarize, a model is therefore typically extended in its measurement method where additional empirical parameters are introduced and analytical cost expressions for collective communication operations are composed using the model to aid the accurate determination of values of primary analytical parameters. One can also say that a model is applied innovatively in this manner. Different application programmers extend the model in different ways based on their intuition, application, and the platform.

The goal of a measurement method, then, is to design an optimal number of reliable experiments that accurately determine the values of the primary analytical parameters by suitably extending the model. Apart from guaranteeing the accuracy, the method must strive to minimize the total execution time of the experiments.

The inputs to a measurement method are usually not mentioned but are quite important to the design of efficient experiments. Some important ones are the following: (a) number of processors; (b) message size; (c) representative communication patterns (involving point-to-point and collective operations); and (d) the nature of the network, for example, switched network, which would allow parallel execution of non-overlapping experiments, thereby reducing the execution time to construct the parameters of a model. Needless to say, the model itself whose parameters are being measured is an input since it is used to derive the analytical cost expressions of point-to-point and collective communications that are used as equations to solve for the model parameters. The output from the measurement method is the values of the primary analytical parameters, where each value is represented by an interval $(\bar{a} \pm \sigma_{\bar{a}})$, where $\bar{a}$ is the estimated value and $\sigma_{\bar{a}}$ is the estimated standard error or standard deviation.

To simplify the exposition, we will assume that there are r analytical parameters of the model. The goal then is to design experiments that construct a linear system of equations $\geq r$. We do not consider models in which the analytical expressions of cost of communications lead to a non-linear system. To compose each equation, one or more experiments are designed and they are repeated multiple times until statistical confidence of the measured values used in the equation (e.g., the execution time) is achieved. The goal, then, is to design the optimal number of experiments to compose the equations and to optimize the number of equations in the linear system.

There are several important guidelines outlined below that should be followed to compose the linear system and to ensure better accuracy of the values of the parameters.

- In theory, to determine the values of the r parameters, one must compose a consistent linear system where the number of linearly independent equations is equal to r. Mathematically speaking, the equations of a linear system are dependent if at least one can be written algebraically as a linear combination of the others. For example, if you have a linear system of three equations where one equation can be algebraically written as a linear combination of the other two equations, then the system is considered linearly dependent. However, the best practice is to compose more than r linearly independent equations and subsequently employ least squares regression to determine the solution. Composing more than r equations should be done for two reasons.
  - The first reason can be attributed to the difficulty in experimentally observing linear dependency in practice. Consider, for example, the construction of Hockney model (Execution time, $T(m) = \alpha + m\,\beta$) with two parameters, α and β, using the input, message size m.

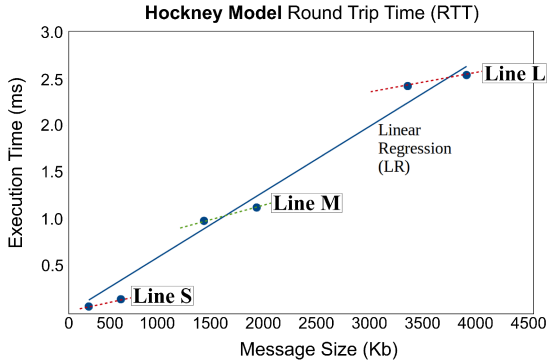

Fig. 11. Illustration of the importance of picking the proper range for an input parameter. In this example, the communication model used is Hockney ($T(m) = \alpha + \beta \times m$) represented by two primary analytical parameters $\{\alpha, \beta\}$ and the input parameter is the message size, m. The three dotted lines, {Line S, Line M, and Line L}, represent linear fits for narrow ranges of small, medium, and large message sizes, respectively. The blue solid line represents linear regression for the overall range. The linear regression fit is $\alpha = 9.73E-07, \beta = 6.783E-07, R^2 = 0.99$.

From the point of view of linear algebra, round-trip experiments with any two different message sizes are sufficient to obtain a system of two independent linear equations, the solution of which will give us the values of α and β. However, depending on selected message sizes, these values can vary significantly. The reason is that real-life experimental points $\{RTT(m_i)\}_{i=1}^{n}$ $(n > 2)$ for any reasonable range of problem sizes never lie on any straight line. Therefore, experiments with only two message sizes are typically not sufficient and the range of values for message size that should be used for composing the equations must be chosen carefully. The advice here is not to use input values too close to each other, which would result in good linear fit for this narrow range but poor fit for other ranges (and overall range). This is illustrated in Figure 11. It shows the measurement of two primary analytical parameters $\{\alpha, \beta\}$ in the Hockney communication model ($T(m) = \alpha + \beta \times m$) using an input parameter, message size (m). The execution times (T) shown are the RTTs between two processes on a real-life cluster. The three dotted lines, {Line S, Line M, and Line L}, represent linear fits for narrow ranges of small, medium, and large message sizes, respectively. The blue solid line is obtained using linear regression for the overall range. One can see that using any one of the three dotted lines can lead to poor prediction accuracy. Hence, one must select a set of message sizes (cardinality more than two) ranging from small to large to compose equations ($> r$) to be input to linear regression for solution. The same applies to other input parameters.

— The second reason relates to improving the ability of the model to predict well for diverse kinds of communication operations. A healthy measurement method must contain at least r logically independent communication experiments. This must be differentiated from linear independence of equations due to practical non-linearity of real-life systems. This means that we need at least r independent equations even under the assumption of perfect linearity of the real-life experimental platform. For example, in the case of the Hockney model, one might consider a round-trip communication experiment involving 3 processors, $p_1 \rightarrow p_2 \rightarrow p_3 \rightarrow p_1$, in addition to the traditional round-trip between two processors for the same message size, m. This additional experiment is not logically independent of round-trips involving two processors as its modeled execution time is

$RTT_3(m) = 3 \times (\alpha + \beta \times m) = \frac{3}{2} \times RTT_2(m)$. Thus, it does not add any new information to the model. On the other hand, a communication experiment comprising a simple collective that involves point-to-point communications between three processors, where one processor communicates with the rest and backwards, will be logically independent from a point-to-point communication involving a pair among the three processors, if the execution time of this experiment is expressed using the *max* operator for parallel branches. In this case, the equation derived from the new experiment extends the model, as the measured execution time is not expressed as a simple linear combination of the execution times of involved point-to-point communications. Therefore, a healthy measurement method must contain at least r logically independent communication experiments, which typically means the use of both point-to-point and collective communication experiments. However, the best practice is to use more than r logically independent experiments for reasons explained below.

- The more the logically independent communication experiments that contain communication operations that are truly representative of the application domain and platform, the better the accuracy. The collectives here must be chosen to allow their intuitive expression and accurate estimation of the model's parameters. For example, Lastovetsky and Rychkov (2007) and Lastovetsky et al. (2009) devise a point-to-two experiment, which is essentially a linear scatter followed by a linear gather. Rico-Gallego et al. (2016) devise an operation $Ring^c_\tau(m)$ to measure the transfer time parameter $L^c(m, \tau)$ for a message size m where they ensure that τ processes transfer data concurrently. It is implemented using MPI_Sendrecv where process P_i sends a message to process P_{i+1}, and receives a message from process P_{i-1} at the same time.
- Since a set of logically independent communication experiments will involve collective operations, the analytical expressions for the collective operations using the model must be employed to frame the equations. This takes care of practical guidelines to be followed in training a communication model, such as use of representative benchmarks to reduce the bias. Since a mix of point-to-point and patterns involving collective operations (representatives from well-known scientific applications) is used in the construction of the model, its prediction accuracy and therefore its ability to predict well for applications employing diverse communication patterns is improved significantly.
- Several issues that arise when experiments employ collectives. First, how should the execution time be measured and at which processor? We propose a collective design such that they commence and terminate at the same (root) processor so that the execution time can be measured at it. For example, consider the case of a linear scatter followed by linear gather between a root processor and the rest. In this case, the clock is started at the root before the commencement of the collective and stopped at the root after the termination of the collective. The difference in clock times gives the execution time of the collective. Second, the collectives can exhibit irregular behaviors such as different slopes in graphs of execution time versus message size for different ranges of message sizes (small and large). For example, Lastovetsky and Rychkov (2007) and Lastovetsky et al. (2009) observe a leap in execution time for linear scatter for large messages on clusters based on a switched network. They introduce a threshold parameter and compose two different sets of equations to account for this irregular behavior, one for message sizes less than or equal to the threshold parameter and the other for sizes greater than the threshold parameter. Therefore, it is imperative to experiment with a wide range of input parameters (such as problem size) and take into account the behavior of collectives specific to a platform.

- If the underlying network is known to be switched, then the communication experiments involving the collectives can be executed in parallel. A good practice is to design the experiments such that they cover all available processors. Therefore, in a set of such experiments, all of the processors participate in more than one experiment, resulting in redundant values of some parameters, which can be used to reduce the number of steps required to achieve statistical significance. This strategy not only reduces the execution time of the measurement method but also improves the accuracy of estimation.
- To make sure that pipelining, cache effects, and the like do not occur, the experiments must not be executed in a loop and sufficient time must be allowed between successive runs of an experiment. This can be achieved, for example, by executing a run in a separate program. For the time between experiments, we propose that it should be based on observations (maximum) of the times taken for memory use to revert to base use and processor (core) frequencies to come back to the base frequencies.
- For reliable estimation of the parameters, the experiments involved in composing each equation must be repeated multiple times until statistical significance of the measured values of parameters used in the equation is achieved. For example, to determine the value of a parameter, the experiments are executed repeatedly until the sample mean lies in the 95% confidence interval and a precision of 0.025 (2.5%) has been achieved. For this purpose, Student's t-test can be used assuming that the individual observations are independent and their population follows the normal distribution, which can be verified using one of the many well-known tests for normality, such as Pearson's chi-squared test.

5 KEY ASPECTS: BUILDING AND REPRESENTATION

A communication performance model aims to provide an accurate estimation of the communication cost of an algorithm and a meaningful analytic representation of the communications in the specific platform in order to aid in making decisions of optimization. We discuss next the factors influencing the performance of a model, namely, the issues in the measurement of the parameters and the influence of representation and design assumptions. The section concludes with a simple reference frame for assessing the overall goodness of a model.

5.1 Accuracy Issues of the Building Procedure

We can identify two main aspects with a decisive influence on the quality of the estimation of the parameters: the tools and procedures used to measure them and their inherent inaccuracy.

5.1.1 Benchmarking Tools. There is a severe shortage of user tools for measuring the parameters of the models, mainly due to the different forms of measuring a parameter in the wide range of existent platforms, with different architectures, middleware, and even operating systems. Hence, the common practice is to resort to micro-benchmarks. This approach involves several subtle platform-dependent issues such as buffer reuse and cache invalidation, synchronization times, and more, which highly influence the final measured value. Note that these laboratory condition factors can be quite different from those present in the field execution environment of an algorithm. Since many portable measurement methods use MPI operations to build a model, it is crucial that the execution time of the MPI operation is determined with precision. Apart from dealing with the various issues, the tools must offer flexible tunable options such as minimum and maximum message size, number of repetitions, and execution time and output a statistical summary.

We will now present a survey of popular benchmarking tools. The Intel MPI Benchmark (IMB) suite (Intel 2004) is a well-known and widely used micro-benchmark. It obtains the execution time, latency, and throughput of MPI point-to-point and collective operations of a given MPI

implementation. It offers several configurable parameters, such as minimum data transfer size, maximum message size, and maximum repetition count for a benchmark.

NetPipe (Turner and Chen 2002) is another popular micro-benchmark. It measures the duration of point-to-point message transmissions following different MPI communication modes as well as the duration of memory copies. Huang et al. (2005) developed a network measurement tool called Hpcbench to measure the latency and throughput of UDP, TCP, and MPI communications over high-performance networks such as Gigabit Ethernet and Myrinet. The tunable parameters are message size, number of repetitions, and allowed execution time of benchmark.

Netgauge (Hoefler et al. 2007) is an extensible framework with a simple interface, which allows the inclusion of new benchmarks for measuring the parameters of newly designed models for a variety of networks. The tool includes the measurements of the parameters of *LogGP* and *PLogP* models. Salnikov et al. (2011) propose a toolkit that allows benchmarking an MPI communication operation and that outputs basic statistics such as mean and standard deviation.

Sound statistical methods should be applied to design these experiments. SKaMPI (Reussner et al. 2002), MPIBench (Grove and Coddington 2001), and MPIBlib (Lastovetsky et al. 2010) are three benchmark suites that employ dispersion metrics along with estimators such as mean, minimum, and maximum values for their MPI measurements. SKaMPI repeatedly executes a benchmark until the current standard error is below an input-allowed threshold. In MPIBench, outliers are treated specifically and detected and removed when their execution time exceeds an input threshold. MPIBlib repeatedly executes a benchmark until the sample mean is within 2.5% of the confidence interval of 95%. The confidence interval of the mean is estimated using Student's t-distribution.

Hamid and Coddington (2010) compare the outputs of several MPI benchmarking tools such as SKaMPI, IMB, and MPIBench and report that they give significantly different results for certain MPI operations, especially on the SGI Altix, with a shared memory ccNUMA architecture. They state that the differences are due to the shortcomings in the tools to account for the architectural peculiarities of SGI Altix and implementation details of SGI MPI for the Altix.

To summarize, the acute shortage of benchmarking tools can be attributed to several factors, the most important being the implementation complexity in supporting diverse architectures, communication models, and the multifarious issues in underlying MPI implementations.

5.1.2 Reproducible Micro-Benchmarking. There is an inherent inaccuracy in parameter measurement. It comes from widely different factors such as system noise or the difficulty in ensuring real-life conditions, as is the case with contention. The point here is that statistical methods must be used for improving accuracy, as mean values over a high number of executions, linear regression methods, addition of communication patterns different from basic point-to-point, and experimental adjustment of the parameter values. Reducing such uncertainty in parameter measurement requires each performance model to specify a precise measurement procedure. A model lacking this procedure is unusable from a practical point of view. The procedure must be flexible enough for adapting it to different platforms. One basic prerequisite that can assist in precise definition of a measurement procedure is an unambiguous definition of the parameters. It also contributes to the understanding of the model representation of the communications.

Almost all the research efforts that we surveyed assume that the measurements/observations are independent and identically distributed. However, such works do not verify that this is the case, assuming that the sample comes from a normally distributed distributionperforming a large number of iterations and taking an arithmetic mean of the observations. Hunold and Carpen-Amarie (2015) describe the common experimental factors (controllable and uncontrollable) typically found in MPI measurement methods. When execution times do not follow a normal distribution, then statistical analysis based on confidence interval for the mean may not be correct. However, if the

sample size is large enough, then the distribution of means will be normal due to the Central Limit Theorem. Based on their experiments using MPI functions for different sample sizes, they recommend use of a large number of samples (3,000 used in their experiments) of size at least 30. Therefore, the question of what is a good number of samples and what is a good size to use that minimizes the execution time of measurements is an open research problem. To make sure that the measurements are independent, they recommend use of the autocorrelation function. It estimates the correlation between two values of a variable measured at different times as a function of the time lag between them. The authors were unable to obtain statistically uncorrelated data even while using random waiting times between a pair of measurements. They recommend the use of sub-sampling to remove correlation.

Hunold and Carpen-Amarie (2016) review the synchronization schemes used in MPI benchmark suites such as MPI_Barrier and window-based using common logical global time and elucidate their strengths and weaknesses. They report that the MPI_Barrier operation may interfere with the communication operation, which is being benchmarked, and that window-based schemes suffer from the issue of distributed clock drift so that the execution time tends to increase with the number of experiments. Their work describes in depth the factors that contribute to a high grade of uncertainty in the measurement.

In addition to the tools used for parameter measurements and the inherent difficulties in their reproducibility, we summarize below a few other factors that can significantly influence the outcome of measurements and that should be factored into the design of MPI experiments.

- The underlying MPI library. For example, while MPICH uses Nemesis for collectives, which are implemented using point-to-point messages, Open MPI provides collectives based on intermediate memory mapped to the processes involved, using its SM component. Hence, different communication mechanisms require different modeling for the same collective operation.
- Different MPI standard communication modes exist. For instance, a buffered send has a different cost than a synchronous send in MPI. Models should consider this issue in their predictions.
- Point-to-point mechanisms such as the segmentation of messages must be taken into account when estimating parameters. Thresholds for segmentation in different libraries could be different. Some libraries may even lack message segmentation.
- The use of internal buffers and cache invalidation in micro-benchmarks is essential to provide accurate estimation. The tools should supply enough configuration facilities in this regard.
- The parameter measurement has to be done under the same conditions as those of algorithms being modeled. This often makes difficult to apply only one set of parameter values to algorithms using different communication mechanisms.

5.2 Design and Representation

Two types of models can be identified depending on the parameter definition and what they represent. First are models with parameters that represent network-related features, as mainstream *LogP/LogGP* and derivatives. They represent the platform behavior with latency, bandwidth, and similar parameters. The main drawbacks of these models come from the representation of complex communication patterns, the increasing complexity of modern HPC platforms, and the wide variety of communication modes.

The first drawback is reflected in the simplicity of analytical representation of collectives. The cost of a collective is usually a function of its number of stages and of the point-to-point cost.

This simplification can lead to mistakes. For instance, the pair of algorithms *Scatter* and *Recursive Doubling Allgather* (RDA) surprisingly results in the same cost formula, as can be seen in Pješivac-Grbović et al. (2007), and was studied in Rico-Gallego and Díaz-Martín (2015). This prediction is a long way from being accurate. It is true that the models get it right that both algorithms share the number of stages ($\log_2 P$). However, they are unable to represent, inter alia, that the whole amount of data moved by RDA exceeds by far that of scatter.

The second drawback comes from the increasing complexity of modern HPC platforms that is driving the growth of new inherited models from *LogP/LogGP*, which add features that reflect specific network or architectural issues, such as *LogGPH* for hierarchical architectures and *LogfP* for Infiniband networks. Contention is also an issue broadly studied. On the whole, however, the efforts to represent contention have produced models too attached to the technological parameters of the target network, aimed at small-sized networks or dealing with nodes with a limited number of cores. The effects of the physical topology of the processors are usually ignored, which contributes to the fact that the contention impact of shared memory is overlooked when tackling collective operations. Furthermore, the models operate in a reduced set of simple collectives. Middleware models provide a more abstract measurement of the contention effects, far from the platform details, and with a good level of accuracy but hiding the underlying causes of performance degradation.

Finally, different communication modes and techniques have been represented in some models. This is the case of *LogGPS* including the synchronization cost of transmissions and *LogGPO* representing the overlap of communication and computation of the communication. Nevertheless, some of the characteristics of the communication are difficult or impossible to represent with network-related parameters. Such characteristics are related to the middleware of communication used in the platform. Middleware implementation includes techniques such as the improvement of the point-to-point communication cost by message segmentation, the design of non-point-to-point collectives on fat shared-memory nodes, communication paradigms not based on point-to-point transmissions such as the *Remote Memory Access* defined in the MPI Standard, operating system bypass to move data directly between different memory spaces in shared memory, or the packing and unpacking of complex data layouts. The influence of these issues is reflected, for instance, in the different costs shown in the execution of the same algorithm by different MPI libraries.

New formal models appear to solve previous issues. The result is a better representation with a possibe loss of accuracy in the estimations and platform details. For instance, $\log_n P$ contributed to the *transfer* concept, the basic building block for representing transmissions. Nevertheless, this model lacks several features commonly found in the current middleware. First, the intra-node performance of MPI is greatly enhanced by segmentation. Even though the segmented and non-segmented transmissions of a message build far different values in the model parameters, segmentation is not explicitly considered in $\log_n P$. It considers a message cost as a half of the actually measured cost in a real point-to-point shared memory transmission. Second, works introducing the model define neither the meaning of contention nor its measurement. In fact, except for elaboration of the parameter, g, contention is not dealt with in this work. Third, although the model is recent, it does not represent the mixture of communications through the different channels present in current systems that is addressed by its inherited counterpart $m\log_n P$. Last, the packing cost is usually not attributable to a point-to-point transmission when it is executed in the context of a collective operation, but the cost is included at the beginning and end of the collective. Beyond some basic broadcast algorithms, there is not enough evidence given today about the competence of $\log_n P$ and $m\log_n P$ to express and predict the cost of the underlying algorithms of the MPI collective operations. τ–Lop departs from the transfer concept and evolves it into a formal

analytic framework able to meaningfully represent the collectives and reach a higher accuracy in the estimations. Nevertheless, although based on a formal and complete definition, its main drawback is the complexity of the parameter measurement procedure and the development of an efficient tool for predicting communication costs based on the parameters.

A further set of issues can be identified in relation to the design assumptions made by the communication models. LogGP, as an example, assumes linearity of the cost of the message transmission with message size. Under LogGP, we can represent with only one parameter the cost of sending a byte G multiplied by the message size. This approach requires precise measurements and statistical methods for approximating the parameter value for accurately predicting the cost of point-to-point transmissions and usually leads to inaccurate predictions of more complex operations, as collectives. Models such as PLogP and $\log_n P$ establish the cost of a message transmission based on parameters that are functions of m and hence contain a vector of scalar values. This approach allows for better accuracy at the expense of a higher number of measurements.

Another assumption is that complex patterns of communication cost are supposed to be influenced only by the number of processes and stages of the algorithm and, hence, scale proportionally to P. Some models include contention representation for partially avoiding this oversimplification, but most also assume that contention grows linearly with the number of transmissions over the specific channel. Furthermore, almost all of the models predict the cost of an algorithm assuming that communication departs from a set of processes ready to start the transmissions at the same time and do not model wait times because of their usual random behavior. Hence, complex applications with multi-pattern communication behavior are nearly impossible to evaluate accurately as a whole. The models do not represent cost as a function of time. As a consequence, current models work better under a *Bulk Synchronous Processing* application model by repeating separate computation and communication stages, with all processes starting the communication stage at the same time. This behavior assumes that the computational load is balanced. While load-balancing strategy used to be a predominant performance optimization strategy before the advent of multicores, it may not give optimal solutions now even in homogeneous environments (Lastovetsky et al. 2015, 2016; Lastovetsky and Reddy 2017).

Finally, general communication models assume a full connectivity network, although some models have been created to represent the cost of more specific networks as meshes and hyper-cubes. Any relaxation of the previous assumptions will improve the usability of a communication performance model.

5.3 A Reference Frame for Model Goodness

Every model is based on a set of parameters whose number, function, and measurement method largely determine its accuracy. In general, few parameters simplify the usage of the model but lead to inaccuracies because of the lack of representation of details involved in the communication. By contrast, a high number of parameters hinder model usability. In this section, we present a reference framework for determining the overall goodness of a model. The framework has three dimensions: Reproducibility, Constructiveness, and Extensibility/Tunability. Table 3 loosely assesses the most notable models based on these factors.

- *Building Reproducibility:* The parameters of a model are defined in abstract terms and are estimated experimentally (built) for each specific platform. Owing to the vast and changing (e.g., homogeneous to heterogeneous) platform landscape, the build procedure needs to be redefined on emerging platforms. The building procedures of a good model would be

Table 3. Salient Properties of Models

Name	Building Reproductibility	Constructiveness	Extensibility/ Tunability
Postal	✔	✔	✘
Hockney	✔	✔	✔
LogP	✔	✔	✔
LogGP	✔	✔	✔
LoPC	✘	✘	✘
Banikazemi	✔	✔	✘
Tam & Wang	✔	✔	✔
Bhat	✔	✔	✔
PLogP	✔	✔	✘
LogGPS	✔	✘	✔
LoGPC	✘	✘	✔
Kim & Lee	✘	✘	✘
HiHCoHP	✘	✔	✘
HLogGP	✘	✘	✘
Steffenel	✔	✔	✔
LogfP	✘	✘	✘
LMO	✔	✔	✔
$\log_n P$	✔	✔	✘
PLP	✘	✔	✘
LogGPO	✘	✔	✔
LogGOPS	✔	✔	✔
LogGPH	✔	✔	✔
$m\log_n P$	✔	✘	✘
Chan	✘	✘	✔
τ-Lop	✔	✔	✔

Note: ✓ and ✗ mean satisfying and not satisfying the property, respectively.

reproducible, and only a comprehensive description of the building procedure ensures its reproducibility.

- *Constructiveness:* The parameters of many models were devised for modeling the cost of point-to-point communication. Its cost expression is then used as a building block of the cost formulations of collective communications. A model that is constructive should be able to build expressive and accurate cost expressions of algorithmic patterns that call point-to-point and collective communications in myriad ways (regular, irregular, unconventional, etc.) We consider that a model satisfies this property if the authors used the model to report prediction accuracy for algorithms that use a mix of point-to-point and collective communications.
- *Extensibility/Tunability:* This means the ability of the model to allow a graceful extension or tuning to improve its practical prediction accuracy for different scenarios. For example, for particular algorithms on specific networks, one may have to introduce extra parameters or formal assumptions to bridge the gap between modeled and experimental results.

6 CONCLUSIONS AND PERSPECTIVES

In this article, we survey prominent communication performance models in the domain of high-performance computing. Our review revealed interesting insights into the evolution of these models.

A communication performance model is conceived to achieve three principal objectives: first, to accurately predict the time of communications; second, to serve as a reliable guide to design and implementation of high-performance computing algorithms; and third, to inspire design of novel high-performance architectures. *BSP* (Valiant 1990) and *LogP* (Culler et al. 1993, 1996a) have been the pioneer models. The notable models that followed in their footsteps have chiefly striven to achieve the first and second objectives. It is also fair to say that the paradigm shifts in HPC have made a noticeable impact on the design path taken by the models. For example, the advent of multi-core architectures resulted in high heterogeneity inside a node. This prompted the conception of middleware performance models such as log_nP (Cameron and Ge 2004; Cameron et al. 2007), $mlog_nP$ (Tu et al. 2012), and τ-*Lop* (Rico-Gallego and Díaz-Martín 2015), which aspire to accurately predict cost of shared-memory transfers and contention inside a node.

Apart from achieving the three main target objectives, a key design goal of a model is to achieve economy in terms of the minimal set of parameters (without sacrificing prediction accuracy) abstracting the essence of communications, which include the costs due to application overhead, latency and bandwidth of a communication network, and the anatomy of a communication network (such as the hierarchical nature, heterogeneity, etc.). Achieving this economy on highly heterogeneous platforms will always be a formidable challenge. The typical pattern when one follows the evolution of models has been to introduce additional parameters to make the models more realistic by accounting for an essential but not-yet-captured characteristic. For example, the *Postal Model* (Bar-Noy and Kipnis 1992) used one parameter to describe the latency of a network. The *Hockney* model (Hockney 1994) followed with two parameters, which described the latency and bandwidth of a network. The *LogP* (Culler et al. 1993, 1996a) model introduced four parameters. The models that followed *LogP* introduced additional parameters accounting for several essential features such as large messages in *LogGP*, contention in *LoPC*, synchronization in *LogGPS*, and so on. A break in the pattern would happen due to an outbreak of a disruptive technology (e.g., the shift toward multi-cores, accelerators, etc.).

In this survey, we also accord commensurate attention to measurement methodologies for determining the values of model parameters and open-source software tools used to build the models. A measurement methodology is evaluated using several criteria: Method, Uncertainty, Overhead, Portability, Intrusiveness, and AfterMath. We conclude that, with few exceptions, the rigor that is followed in presenting a model is usually missing when presenting a measurement methodology. There are several open-source software tools that have been developed to automate the construction of models. This software automation effort is made remarkably difficult by the myriad and complex nature of optimizations in the underlying implementation (e.g., the MPI implementation). In addition, the disruptive shifts in HPC landscape (e.g., advent of multi-cores, accelerators, etc.) means that the tools can quickly become outdated unless they keep abreast with technological advancements.

Looking into the future, we envisage three areas that will have a salient impact on longevity of existing models and the design and implementation of novel communication performance models. First, energy is now a leading design constraint along with performance in the HPC domain. There is an abysmal lack of realistic and accurate energy models of communications. We believe that a holistic approach employing realistic and accurate performance and energy models of computations and communications is crucial for optimization of scientific applications on modern HPC platforms for both performance and energy. Second, the advent of software-defined networks (SDNs), which afford users high software configurability and programmability of networks, may elicit design of new models with novel abstractions. With several users simultaneously tuning (configuring) the network to suit the needs of their applications, the number of degrees of freedom will increase drastically. The question, then, is how will the models be able to deal with

this additional complexity to accurately predict the cost of communications? Finally, there is now a zealous focus on exascale computing. This may pose new challenges to the existing models and may even necessitate a fresh/radical approach to design of novel models. We envision two major challenges. The first is how to deal with the behavior of the model parameters with respect to message size and the number of processors involved in the execution of the communication operation as scale grows from small to extremely large. The question is, will the model be able to predict accurately several irregularities or non-linearities in execution time of communications with increase in scale? The second challenge is how to use the existing models innovatively to optimize communications at this scale. A hierarchical approach is one possible way forward.

REFERENCES

A. Aggarwal, B. Alpern, A. Chandra, and M. Snir. 1987. A model for hierarchical memory. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC'87)*. ACM, New York, NY, 305–314.

Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. 1995. LogGP: Incorporating long messages into the LogP model - one step closer towards a realistic model for parallel computation. In *Proceedings of the 7th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'95)*. New York, NY, 95–105.

M. Banikazemi, V. Moorthy, and D. K. Panda. 1998. Efficient collective communication on heterogeneous networks of workstations. In *Proceedings of the International Conference on Parallel Processing*. 460–467.

M. Banikazemi, J. Sampathkumar, S. Prabhu, D. K. Panda, and P. Sadayappan. 1999. Communication modeling of heterogeneous networks of workstations for performance characterization of collective operations. In *Proceedings of the 8th Heterogeneous Computing Workshop (HCW'99)*. 125–133.

Amotz Bar-Noy and Shlomo Kipnis. 1992. Designing broadcasting algorithms in the postal model for message-passing systems. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'92)*. ACM, New York, NY, 13–22.

LuizAngelo Barchet-Estefanel and Grégory Mounié. 2004. *Fast Tuning of Intra-cluster Collective Communications*. Lecture Notes in Computer Science, Vol. 3241. Springer, Berlin, 28–35.

LuizAngelo Barchet-Steffenel and Grégory Mounié. 2006. *Total Exchange Performance Modelling Under Network Contention*. Lecture Notes in Computer Science, Vol. 3911. Springer, Berlin, 100–107.

Massimo Bernaschi and Giulio Iannello. 1998. Collective communication operations: Experimental results vs. theory. *Concurrency: Practice and Experience* 10, 5 (4 1998), 359–386.

Prashanth B. Bhat, Viktor K. Prasanna, and C. S. Raghavendra. 1999. Adaptive communication algorithms for distributed heterogeneous systems. *Journal of Parallel and Distributed Computing* 59, 2 (Nov. 1999), 252–279.

Prashanth B. Bhat, C. S. Raghavendra, and Viktor K. Prasanna. 2003. Efficient collective communication in distributed heterogeneous systems. *Journal of Parallel and Distributed Computing* 63, 3 (March 2003), 251–263.

J. L. Bosque and L. Pastor. 2006. A parallel computational model for heterogeneous clusters. *IEEE Transactions on Parallel and Distributed Systems* 17 (2006), 1390–1400.

J. L. Bosque and L. P. Perez. 2004. HLogGP: A new parallel computational model for heterogeneous clusters. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid'04)*. IEEE, 403–410.

Kirk W. Cameron and Rong Ge. 2004. Predicting and evaluating distributed communication performance. In *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing (SC'04)*. IEEE Computer Society, Washington, DC, 43.

K. W. Cameron, R. Ge, and X. H. Sun. 2007. $\log_m P$ and $\log_3 P$: Accurate analytical models of point-to-point communication in distributed systems. *IEEE Transactions on Computers* 56, 3 (2007), 314–327.

Kirk W. Cameron and Xian-He Sun. 2003. Quantifying locality effect in data access delay: Memory logP. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing (IPDPS'03)*. IEEE Computer Society, Washington, DC, 48.2.

F. Cappello, P. Fraigniaud, B. Mans, and A. L. Rosenberg. 2001. HiHCoHP—Toward a realistic communication model for hierarchical hyperclusters of heterogeneous processors. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium*. IEEE, 6.

Franck Cappello, Pierre Fraigniaud, Bernard Mans, and Arnold L. Rosenberg. 2005. An algorithmic model for heterogeneous hyper-clusters: Rationale and experience. *International Journal of Foundations of Computer Science* 16, 2 (2005), 195–215.

Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. 2014. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing* 74, 10 (June 2014), 2899–2917.

Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert van de Geijn. 2007. Collective communication: Theory, practice, and experience: Research articles. *Concurrency and Computation: Practical Experience* 19, 13 (Sept. 2007), 1749–1783.

Siew Yin Chan, Teck Chaw Ling, and Eric Aubanel. 2015. Performance modeling for hierarchical graph partitioning in heterogeneous multi-core environment. *Parallel Computing* 46, C (July 2015), 78–97.

WenGuang Chen, JiDong Zhai, Jin Zhang, and WeiMin Zheng. 2009. LogGPO: An accurate communication model for performance prediction of MPI programs. *Science in China Series F: Information Sciences* 52, 10 (2009), 1785–1791.

Chau-Yi Chou, Hsi-Ya Chang, Shuen-Tai Wang, Kuo-Chan Huang, and Cherng-Yeu Shen. 2007. An improved model for predicting HPL performance. In *Advances in Grid and Pervasive Computing*, Christophe Cérin and Kuan-Ching Li (Eds.). Springer, Berlin, 158–168.

David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. 1993. LogP: Towards a realistic model of parallel computation. In *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP'93)*. ACM, New York, NY, 1–12.

David E. Culler, Richard M. Karp, David Patterson, Abhijit Sahay, Eunice E. Santos, Klaus Erik Schauser, Ramesh Subramonian, and Thorsten von Eicken. 1996a. LogP: A practical model of parallel computation. *Communications of the ACM* 39, 11 (November 1996), 78–85.

David E. Culler, Lok Tin Liu, Richard P. Martin, and Chad O. Yoshikawa. 1996b. Assessing fast network interfaces. *IEEE Micro* 16, 1 (Feb. 1996), 35–43.

Jack Dongarra, Pete Beckman, Terry Moore, Patrick Aerts, Giovanni Aloisio, Jean-Claude Andre, David Barkai, Jean-Yves Berthou, Taisuke Boku, Bertrand Braunschweig, Franck Cappello, Barbara Chapman, Xuebin Chi, Alok Choudhary, Sudip Dosanjh, Thom Dunning, Sandro Fiore, Al Geist, Bill Gropp, Robert Harrison, Mark Hereld, Michael Heroux, Adolfy Hoisie, Koh Hotta, Zhong Jin, Yutaka Ishikawa, Fred Johnson, Sanjay Kale, Richard Kenway, David Keyes, Bill Kramer, Jesus Labarta, Alain Lichnewsky, Thomas Lippert, Bob Lucas, Barney Maccabe, Satoshi Matsuoka, Paul Messina, Peter Michielse, Bernd Mohr, Matthias S. Mueller, Wolfgang E. Nagel, Hiroshi Nakashima, Michael E. Papka, Dan Reed, Mitsuhisa Sato, Ed Seidel, John Shalf, David Skinner, Marc Snir, Thomas Sterling, Rick Stevens, Fred Streitz, Bob Sugar, Shinji Sumimoto, William Tang, John Taylor, Rajeev Thakur, Anne Trefethen, Mateo Valero, Aad Van Der Steen, Jeffrey Vetter, Peg Williams, Robert Wisniewski, and Kathy Yelick. 2011. The International Exascale Software Project Roadmap. *International Journal of High Performance Computing Applications* 25, 1 (Feb. 2011), 3–60.

Andrea C. Dusseau, David E. Culler, Klaus Erik Schauser, and Richard P. Martin. 1996. Fast parallel sorting under LogP: Experience with the CM-5. *IEEE Transactions on Parallel Distributed Systems* 7, 8 (Aug. 1996), 791–805.

Matthew I. Frank, Anant Agarwal, and Mary K. Vernon. 1997. LoPC: Modeling contention in parallel algorithms. In *Proceedings of the 6th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP'97)*. ACM, New York, NY, 276–287.

Duncan Grove and Paul Coddington. 2001. Precise MPI performance measurement using MPIBench. In *Proceedings of HPC Asia*. ACM.

Nor Asilah Wati Abdul Hamid and Paul Coddington. 2010. Comparison of MPI benchmark programs on shared memory and distributed memory machines (point-to-point communication). *International Journal of High Performance Computing Applications* 24, 4 (2010), 469–483.

Khalid Hasanov and Alexey Lastovetsky. 2017. Hierarchical redesign of classic MPI reduction algorithms. *Journal of Supercomputing* 73 (02/2017 2017), 1–13.

Khalid Hasanov, Jean-Noel Quintin, and Alexey Lastovetsky. 2015a. Hierarchical approach to optimization of parallel matrix multiplication on large-scale platforms. *Journal of Supercomputing* 71 (11/2015 2015), 3991–4014.

Khalid Hasanov, Jean-Noel Quintin, and Alexey Lastovetsky. 2015b. Topology-oblivious optimization of MPI broadcast algorithms on extreme-scale platforms. *Simulation Modelling Practice and Theory* 58 (11/2015 2015), 30–39.

J. Hatta and S. Shibusawa. 2000. Scheduling algorithms for efficient gather operations in distributed heterogeneous systems. In *Proceedings of the International Workshops on Parallel Processing*. IEEE Computer Society, 173–180.

Roger W. Hockney. 1994. The communication challenge for MPP: Intel paragon and Meiko CS-2. *Parallel Computing* 20, 3 (March 1994), 389–398.

Torsten Hoefler, Lavinio Cerquetti, and Frank Mietke. 2005. A practical approach to the rating of barrier algorithms using the LogP model and open MPI. In *Proceedings of the International Conference on Parallel Processing Workshops (ICPPW'05)*. IEEE Computer Society, Washington, DC, 562–569.

Torsten Hoefler, Torsten Mehlan, Andrew Lumsdaine, and Wolfgang Rehm. 2007. Netgauge: A network performance measurement framework. In *Proceedings of the 3rd International Conference on High Performance Computing and Communications (HPCC'07)*. Springer, Berlin, 659–671.

T. Hoefler, T. Mehlan, F. Mietke, and Wolfgang Rehm. 2006. LogfP - A model for small messages in InfiniBand. In *20th International Parallel and Distributed Processing Symposium (IPDPS'06)*. IEEE Computer Society, 319.

T. Hoefler and W. Rehm. 2005. A communication model for small messages with InfiniBand. In *PARS Mitteilungen*. PARS, 32–41.

T. Hoefler, T. Schneider, and A. Lumsdaine. 2008. Multistage switches are not crossbars: Effects of static routing in high-performance networks. In *IEEE International Conference on Cluster Computing*. IEEE, 116–125.

T. Hoefler, T. Schneider, and A. Lumsdaine. 2009a. LogGP in theory and practice: An in-depth analysis of modern interconnection networks and benchmarking methods for collective operations. *Simulation Modelling Practice and Theory* 17, 9 (2009), 1511–1521.

Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. 2010. LogGOPSim: Simulating large-scale applications in the LogGOPS model. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC'10)*. ACM, New York, NY, 597–604.

Torsten Hoefler, Christian Siebert, and Andrew Lumsdaine. 2009b. Group operation assembly language - A flexible way to express collective communication. In *Proceedings of the International Conference on Parallel Processing (ICPP'09)*. IEEE Computer Society, Washington, DC, 574–581.

Chris Holt, Mark Heinrich, Jaswinder P. Singh, Edward Rothberg, and John Hennessy. 1995. *The Effects of Latency, Occupancy, and Bandwidth in Distributed Shared Memory Multiprocessors*. Technical Report. Stanford University, Stanford, CA.

B. Huang, M. Bauer, and M. Katchabaw. 2005. Hpcbench - A Linux-based network benchmark for high performance networks. In *19th International Symposium on High Performance Computing Systems and Applications (HPCS'05)*. IEEE Computer Society, 65–71.

Sascha Hunold and Alexandra Carpen-Amarie. 2015. MPI benchmarking revisited: Experimental design and reproducibility. *CoRR (ArXiv)* abs/1505.07734.

Sascha Hunold and Alexandra Carpen-Amarie. 2016. Reproducible MPI benchmarking is still not as easy as you think. *IEEE Transactions on Parallel Distributed Systems* 27, 12 (Dec. 2016), 3617–3630.

G. Iannello, M. Lauria, and S. Mercolino. 1998. LogP performance characterization of fast messages atop Myrinet. In *Proceedings of the 6th Euromicro Workshop on Parallel and Distributed Processing (PDP'98)*. IEEE, 395–401.

Fumihiko Ino, Noriyuki Fujimoto, and Kenichi Hagihara. 2001. LogGPS: A parallel computational model for synchronization analysis. *SIGPLAN Notes* 36, 7 (June 2001), 133–142.

Intel. 2004. Intel MPI Benchmarks. Users Guide and Methodology Description. https://software.intel.com/en-us/imb-user-guide.

Joseph JáJá. 1992. *An Introduction to Parallel Algorithms*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA.

V. Jerome, M. Maxime, V. Jean-Marc, and M. Jean-Francois. 2008. Predictive models for bandwidth sharing in high performance clusters. In *IEEE International Conference on Cluster Computing*. IEEE, 286–291.

L. V. Kale, S. Kumar, and K. Varadarajan. 2003. A framework for collective personalized communication. In *Proceedings of the International Parallel and Distributed Processing Symposium*. IEEE Computer Society, 9.

Richard M. Karp, Abhijit Sahay, Eunice E. Santos, and Klaus Erik Schauser. 1993. Optimal broadcast and summation in the LogP model. In *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'93)*. ACM, New York, NY, 142–153.

Thilo Kielmann, Henri E. Bal, Sergei Gorlatch, Kees Verstoep, and Rutger F. H. Hofman. 2001. Network performance-aware collective communication for clustered wide-area systems. *Parallel Computing* 27, 11 (2001), 1431–1456. https://doi.org/10.1016/S0167-8191(01)00098-9

Thilo Kielmann, Henri E. Bal, and Kees Verstoep. 2000. Fast measurement of LogP parameters for message passing platforms. In *Proceedings of the 15 IPDPS Workshops on Parallel and Distributed Processing (IPDPS'00)*. Springer, London, UK, 1176–1183.

Sang Cheol Kim and Sunggu Lee. 2001. Measurement and prediction of communication delays in Myrinet networks. *Journal of Parallel and Distributed Computing* 61, 11 (Nov. 2001), 1692–1704.

Alexey Lastovetsky. 2002. Adaptive parallel computing on heterogeneous networks with mpC. *Parallel Computing* 28, 10 (Oct. 2002), 1369–1407.

A. Lastovetsky, I.-H. Mkwawa, and M. O'Flynn. 2006. An accurate communication model of a heterogeneous cluster based on a switch-enabled Ethernet network. In *12th International Conference on Parallel and Distributed Systems (ICPADS'06)*. Vol. 2. IEEE Computer Society, 6.

A. Lastovetsky and R. Reddy. 2017. New model-based methods and algorithms for performance and energy optimization of data parallel applications on homogeneous multicore clusters. *IEEE Transactions on Parallel and Distributed Systems* 28, 4 (April 2017), 1119–1133.

A. Lastovetsky and V. Rychkov. 2007. Building the communication performance model of heterogeneous clusters based on a switched network. In *IEEE International Conference on Cluster Computing*. IEEE, 568–575.

A. Lastovetsky and V. Rychkov. 2009. Accurate estimation of parameters of heterogeneous communication performance models. *International Journal of High Performance Computing Applications* 23, 2 (May 2009), 123–139.

A. Lastovetsky, V. Rychkov, and M. O'Flynn. 2009. Revisiting communication performance models for computational clusters. In *IEEE International Symposium on Parallel Distributed Processing (IPDPS'09)*. IEEE, 1–11.

A. Lastovetsky, V. Rychkov, and M. O'Flynn. 2010. Accurate heterogeneous communication models and a software tool for their efficient estimation. *International Journal of High Performance Computing Applications* 24, 1 (Feb. 2010), 34–48.

A. Lastovetsky, L. Szustak, and R. Wyrzykowski. 2015. Model-based optimization of MPDATA on Intel Xeon Phi through load imbalancing. *CoRR (ArXiv)* abs/1507.01265. http://arxiv.org/abs/1507.01265

A. Lastovetsky, L. Szustak, and R. Wyrzykowski. 2016. Model-based optimization of EULAG kernel on Intel Xeon Phi through load imbalancing. *IEEE Transactions on Parallel and Distributed Systems* 28, 3 (2016), 787–797.

B. M. Maggs, L. R. Matheson, and R. E. Tarjan. 1995. Models of parallel computation: A survey and synthesis. In *Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS'95)*. IEEE Computer Society, Washington, DC, 61–71.

Richard P. Martin, Amin M. Vahdat, David E. Culler, and Thomas E. Anderson. 1997. Effects of communication latency, overhead, and bandwidth in a cluster architecture. *SIGARCH Computer Architecture News* 25, 2 (May 1997), 85–97.

Maxime Martinasso and Jean-François Méhaut. 2005. *Prediction of Communication Latency over Complex Network Behaviors on SMP Clusters*. Lecture Notes in Computer Science, Vol. 3670. Springer, Berlin, 172–186.

Maxime Martinasso and Jean-François Méhaut. 2011. A contention-aware performance model for HPC-based networks: A case study of the InfiniBand network. In *Proceedings of the 17th International Conference on Parallel Processing — Volume Part I (Euro-Par'11)*. Springer, Berlin, 91–102.

Csaba Andras Moritz and Matthew I. Frank. 2001. LoGPC: Modeling network contention in message-passing programs. *IEEE Transactions on Parallel and Distributed Systems* 12, 4 (April 2001), 404–415.

MPI Forum. 2012. MPI: A Message-Passing Interface Standard. Version 3.0. Retrieved December 15, 2018 from http://www.mpi-forum.org.

W. Nasri, O. Tarhouni, and N. Slimi. 2008. PLP: Towards a realistic and accurate model for communication performances on hierarchical cluster-based systems. In *IEEE International Symposium on Parallel and Distributed Processing (IPDPS'08)*. IEEE, 1–8.

F. Ooshita, S. Matsumae, and T. Masuzawa. 2002. Efficient gather operation in heterogeneous cluster systems. In *Proceedings of the 16th Annual International Symposium on High Performance Computing Systems and Applications*. 196–204.

Ju-Young L. Park, Hyeong-Ah Choi, Natawut Nupairoj, and L. M. Ni. 1996. Construction of optimal multicast trees based on the parameterized communication model. In *Proceedings of the ICPP Workshop on Challenges for Parallel Processing*, Vol. 1. IEEE, 180–187. https://doi.org/10.1109/ICPP.1996.537159

A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary. 2016. HPL Scalability Analysis. Retrieved December 15, 2018 from http://www.netlib.org/benchmark/hpl/scalability.html.

Jelena Pješivac-Grbović, Thara Angskun, George Bosilca, Graham E. Fagg, Edgar Gabriel, and Jack J. Dongarra. 2007. Performance analysis of MPI collective operations. *Cluster Computing* 10, 2 (June 2007), 127–143.

MPICH. 2013. Argonne National Laboratory. Retrieved December 15, 2018 from https://www.mpich.org.

Rolf Rabenseifner and Jesper Larsson Träff. 2004. *More Efficient Reduction Algorithms for Non-Power-of-Two Number of Processors in Message-Passing Parallel Systems*. Lecture Notes in Computer Science, Vol. 3241. Springer, Berlin, 36–46.

Ralf Reussner, Peter Sanders, and Jesper Larsson Träff. 2002. SKaMPI: A comprehensive benchmark for public benchmarking of MPI. *Scientific Programming* 10, 1 (Jan. 2002), 55–65.

J. A. Rico-Gallego, J. C. Díaz-Martín, and A. L. Lastovetsky. 2016. Extending τ–Lop to model concurrent MPI communications in multicore clusters. *Future Generation Computer Systems* 61 (2016), 66–82.

J. A. Rico-Gallego and J. C. Díaz-Martín. 2015. τ–Lop: Modeling performance of shared memory MPI. *Parallel Computing* 46 (2015), 14–31.

J. A. Rico-Gallego, A. L. Lastovetsky, and J. C. Díaz-Martín. 2017. Model-based estimation of the communication cost of hybrid data-parallel applications on heterogeneous clusters. *IEEE Transactions on Parallel and Distributed Systems* 28, 11 (Nov 2017), 3215–3228.

Jeff Rothenberg, Lawrence E. Widman, Kenneth A. Loparo, and Norman R. Nielsen. 1989. The nature of modeling. In *Artificial Intelligence, Simulation and Modeling*. John Wiley and Sons, 75–92.

Alexey Salnikov, Dmitry Andreev, and Roman Lebedev. 2011. The analysis of cluster interconnect with the network_tests2 toolkit. In *Recent Advances in the Message Passing Interface*. Springer, Berlin, 160–169.

Eunice E. Santos. 1999. Optimal and near-optimal algorithms fork-item broadcast. *Journal of Parallel and Distributed Computing* 57, 2 (1999), 121–139.

L. A. Steffenel. 2006. Modeling network contention effects on all-to-all operations. In *IEEE International Conference on Cluster Computing*. IEEE, 1–10.

A. T. C. Tam and Cho-Li Wang. 1999. Realistic communication model for parallel computing on cluster. In *Proceedings of the 1st IEEE Computer Society International Workshop on Cluster Computing*. IEEE, 92–101.

A. T. C. Tam and Cho-Li Wang. 2003. Contention-aware communication schedule for high-speed communication. *Cluster Computing* 6, 4 (2003), 339–353.

Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of collective communication operations in MPICH. *International Journal of High Performance Computing Applications* 19, 1 (2005), 49–66.

Top500. 2018. TOP500—The List. Retrieved December 15, 2018 from https://www.top500.org/lists/2018/06/.

Jesper Larsson Träff and Andreas Ripke. 2005. *An Optimal Broadcast Algorithm Adapted to SMP Clusters*. Lecture Notes in Computer Science, Vol. 3666. Springer, Berlin, 48–56.

Bibo Tu, Jianping Fan, Jianfeng Zhan, and Xiaofang Zhao. 2012. Performance analysis and optimization of MPI collective operations on multi-core clusters. *Journal of Supercomputing* 60, 1 (2012), 141–162.

D. Turner and Xuehua Chen. 2002. Protocol-dependent message-passing performance on Linux clusters. In *Proceedings of the IEEE International Conference on Cluster Computing*. IEEE, 187–194.

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

R. A. Van De Geijn and J. Watts. 1997. SUMMA: Scalable universal matrix multiplication algorithm. *Concurrency: Practice and Experience* 9, 4 (1997), 255–274.

Liang Yuan, Yunquan Zhang, Yuxin Tang, Li Rao, and Xiangzheng Sun. 2010. LogGPH: A parallel computational model with hierarchical communication awareness. In *IEEE 13th International Conference on Computational Science and Engineering (CSE'10)*. IEEE, 268–274.

Jun Zhu, Alexey Lastovetsky, Shoukat Ali, and Rolf Riesen. 2013. Communication models for resource constrained hierarchical ethernet networks. In *11th International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar'13)*. Lecture Notes in Computer Science, Vol. 8374, Springer.

Received December 2017; revised August 2018; accepted September 2018