A sampling-based approach for communication libraries auto-tuning

Elisabeth Brunet, François Trahay, Alexandre Denis, Raymond Namyst

A sampling-based approach for communication libraries auto-tuning

Élisabeth Brunet, François Trahay
Institut Télécom, Télécom SudParis, Evry, France
Elisabeth.Brunet@it-sudparis.eu
Francois.Trahay@it-sudparis.eu

Alexandre Denis
INRIA Bordeaux – Sud-Ouest / LaBRI, France
Alexandre.Denis@inria.fr

Raymond Namyst
University of Bordeaux / LaBRI, France
Raymond.Namyst@labri.fr

***Abstract*—Communication performance is a critical issue in HPC applications, and many solutions have been proposed on the literature (algorithmic, protocols, etc.) In the meantime, computing nodes become massively multicore, leading to a real imbalance between the number of communication sources and the number of physical communication resources. Thus it is now mandatory to share network boards between computation flows, and to take this sharing into account while performing communication optimizations.**

In previous papers, we have proposed a model and a framework for on-the-fly optimizations of multiplexed concurrent communication flows, and implemented this model in the NEWMADELEINE communication library. This library features optimization strategies able for example to aggregate several messages to reduce the number of packets emitted on the network, or to split messages to use several NICs at the same time.

In this paper, we study the tuning of these dynamic optimization strategies. We show that some parameters and thresholds (*rendezvous* threshold, aggregation packet size) depend on the actual hardware, both host and NICs. We propose and implement a method based on sampling of the actual hardware to auto-tune our strategies. Moreover, we show that multi-rail can greatly benefit from performance predictions. We propose an approach for multi-rail that dynamically balance the data between NICs using predictions based on sampling.

I. INTRODUCTION

The success of cluster architectures as the most widespread platform for high performance computing comes from the aggressive performance/cost ratio. A challenge in exploiting such architectures is to hide the communication cost. As low-level drivers are rather difficult to efficiently exploit, intermediate communication libraries propose to abstract the use of the most widespread network technologies and have to know precisely the drivers' behavior to harness the full capacities. Several transfer methods are usually required in order to achieve optimal transfer whatever the message size – for example, in message passing paradigm context, small messages use to be sent through a raw transfer while larger ones are managed after a handshake between both communication sides, etc. In that case, the selection of the adequate transfer method, related to message sizes, is usually defined by hard-coded thresholds that rely on the driver's documentation or on an empirical study made by the programmer. However, the efficiency of the underlying transfer methods highly depends on the characteristics of the machines that actually run the application. The hardware type, brand, low-level driver version, for the NIC and even the host configuration are all determining parameters.

The introduction of multicore processors in clusters leads to other challenges for communication libraries as the number of communication sources increases much faster than the network resources. The NICs thus have to be shared by multiple processing units. The need for multiplexing as well as the increase of communication leads the way to optimization opportunities. The performance of each of those potential optimizations (such as aggregation, reordering, splitting) is affected by various aspects of the machines that run the application. Collecting information on the performance of each strategy permits to apply the most appropriate.

In this paper, we study the impact of various characteristics of the machine on communication achievements. We propose and implement in NEWMADELEINE, our communication library, a method based on sampling of the actual hardware to auto-tune the optimization strategies. Section 2 presents the hardware features that we focus on and their impact on our strategies. The measurement collection as well as its use by the strategies in NEWMADELEINE is explained in Section 3. The results of the evaluation of the sampling-based system are presented in Section 4. Related works are described in Section 5 and Section 6 draws a conclusion and introduces about future works.

II. SAMPLING-BASED AUTO-TUNING OF A COMMUNICATION LIBRARY

This section shows different types of optimizations, ranging from low-level mechanisms to high-level message reordering, which demonstrate utility in harnessing the full capacities of network technologies.

A. Methodology overview

Communication libraries aim at achieving the best possible performance with a given network hardware. Most

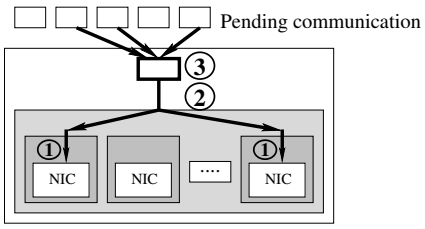

Figure 1. Levels of tunable network mechanisms: (1) protocol: to perform a single communication on a given network; (2) multi-rail: to balance over multiple networks; (3) optimization: to apply on-the-fly optimization scheduling strategies.

modern communication libraries use complex protocols and optimization mechanisms. Their efficiency depends on the actual network and host hardware, making their tuning a very difficult task. In this paper, we distinguish three levels of mechanisms, as depicted in Figure 1:

- **protocol** — To optimize a single raw communication on a given network, communication libraries usually implement several communication transfer methods, depending on the message size, with thresholds to switch from one method to another. However, these thresholds are likely to vary between hardware type, brand, and software configuration.
- **multi-rail** — On clusters with multiple NICs per node, the communication library has to arbitrate the access to the physical communication resources, in order to interleave communication flows and make communication progress fairly. Balancing packets between NICs requires to predict accurately their performance.
- **optimization** — Some communication libraries are able to apply optimization patterns on the fly in various ways (aggregation, reordering, etc.) depending on the available physical resources. Knowing the costs and benefits of these operations allows the library to apply the most effective optimization operations to packets.

It is possible to use *quirks* — choose among a list of known network hardware with empirically determined values for all parameters and thresholds —, but this method is highly non-portable and difficult to maintain in the long term. Moreover, the network card is not the only parameter. The CPU and memory bandwidth have a major impact [1] on communication performance. Even further, driver version, host configuration, operating system, and `libc` version (`memcpy` implementation vary from one version to another) may have an influence on communication transfer.

Given the large number of parameters and their interactions, we propose to work with real information by measuring the actual performance of the communication library on a wide spectrum of parameters (eg. message size, performance of memcpy, etc.) Then, these results are used by the library to compute the best value for all parameters and to predict the performance of the network.

B. Adapting transfer methods choice to the hardware

Communication libraries usually implement at least two different communication methods, known as *eager* and *rendezvous*. When sending data eagerly, the sender does not know whether the receiver is ready or not, i.e. whether the application has posted its `recv` or not. Data is copied in memory at the receiver side in the case of an unexpected message. However, memory copies decrease bandwidth for large messages. To avoid such an impact on bandwidth, communication libraries usually implement a *rendezvous* mechanism: the sender sends a *rendezvous* request; the receiver replies when it is ready to receive; finally the sender sends data and the receiver receives it without any copy.

Compared to raw network transfer, the overhead of the *eager* method is a memory copy; the overhead of the *rendezvous* method is round-trip time. *eager* mode is used for small messages, and *rendezvous* for large messages. The threshold to switch from *eager* to *rendezvous* is the size where the overhead of the memory copy becomes higher than the cost of a round-trip.

However, these overheads highly depend on the hardware, namely the performance profile of memory copies and the performance of the network. It is expected that no fixed value will fit any random hardware.

At a lower level, communication libraries need *headers* for small packets to describe their content, such as the message tag, communicator ID, or sequence number. Moreover, most modern communication libraries support `iovec` from the application. A packet is therefore comprised of multiple parts scattered in memory. Two approaches are possible to assemble them into a packet ready to be sent on the network: a copy to flatten all parts, header and body, into a contiguous block; or give directly an `iovec` to the network driver.

When a message is constructed incrementally, it is often beneficial to copy early during the packet construction, with a copy taking place in the L1 cache. However, most network drivers actually perform themselves a memory copy for small messages, in order to place data into pre-registered memory. The driver is able to flatten an `iovec` into contiguous memory on the fly. The compromise between copying scattered data into a contiguous buffer early twice, or late once, depends on the memory performance profile, the network hardware, and the network driver. Our approach to choose the best method between a copy in the communication library or to give an `iovec` to the driver is to sample both methods, and to decide from the actual performance.

C. Tuning multi-rail to actual hardware

The massive adoption of multicore processors creates numerous concurrent communication flows. To manage this

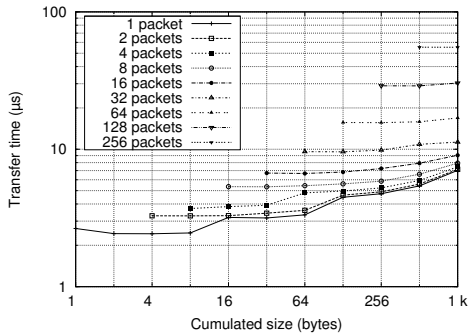

Figure 2. Subdivision of a given length in various numbers of messages — Latency.

increase, it is now not uncommon to have multiple NICs per node.

A natural way to use multiple NICs consists in a greedy scheduling where a message is scheduled on the first available NIC. When several NICs are available, it is beneficial to use all available NICS by splitting packets into as many chunks as NICs. In the case where not all NICs are the same, the heterogeneous hardware results in heterogeneous performance. Knowing precisely the network performance permits to adapt the *split ratio*: send a larger chunk of packet through the fastest NIC so that the transmission of both chunks complete at the same time.

Furthermore, a greedy strategy based upon immediate network availability is not always the best. For example, instead of sending immediately on a slow NIC, it is worth delaying the transfer of a large packet to wait for a faster NIC predicted to become available very soon. If the performance of the underlying network drivers is known precisely, the communication library is able to predict when a NIC will become idle. In the case of multi-rail, this ability to predict the NIC behavior permits to compute the optimal scheduling for message transfers.

D. Sampling-based optimization strategy

When applications have a complex communication scheme, it is possible for the communication libraries to apply on the fly some optimization strategies, such as aggregation or reordering. The aggregation-based optimization relies on the fact that sending two packets of a given size takes more time than sending a single packet twice as large. Therefore, for a certain set of messages, it is worth aggregating them into a single packet rather than sending them separately on the network. It may be generalized to more than two packets, as shown in Figure 2. The time difference is in the order of magnitude of the network latency. Thus it is worth aggregating multiple packets before sending them on the network. However, since aggregation needs a memory copy to copy packets contiguously, for large messages memory copy may be more expensive than the expected gain of aggregation.

For a given queue of pending messages to send, if performance may be predicted through sampling, then finding the best scheduling that gives the lowest transfer time is a combinatorial optimization problem that may be solved through complete exploration. However this problem is NP-complete (reduces to the knapsack problem). Since we have to apply the optimization strategy on the communication critical path, where we may at most spend computations in the order of $100\,ns$, we have proposed [2] a very simple heuristic based on the size of messages. Since we cannot predict the future messages that the application will send, we restrict ourselves to the worst case, which is the aggregation of two packets. Given two packets, it may be determined through sampling whether it is worth aggregating them or not.

III. Implementation of strategies of optimization auto-tuning in NewMadeleine

We have implemented the sampling-based auto-tuned optimization strategies in NewMadeleine [2], our communication library which follows the NICs activity. NewMadeleine behavior is totally untied from the application: while the NICs are used, it stores applicative communication requests; when a NIC is released, it combines those pending requests through optimization strategies before feeding the network. Various strategies can be applied such as aggregation, reordering or load balancing across several NICs. In all these cases, using one strategy instead of another or determining when an optimization scope becomes obsolete from a given threshold depends on the machine characteristics. Thus, NewMadeleine strategies have been tuned with a sampling-based toolbox.

In this Section we present how the information is collected and afterward supplied to NewMadeleine protocols and strategies. We then detail how the aggregation and multi-rail strategies work.

A. Collecting the samples

NewMadeleine profiles the actual hardware at initialization before the application is launched. Profiling is performed once for a given platform, and those measurements are persistent across runs on the same platform. Several kinds of benchmarks are performed. Each transfer method proposed by the NewMadeleine driver is sampled with a ping-pong test on a large spectrum of message sizes. These samples are then used for auto-tuning in two ways:

- the sampling program computes *thresholds* between methods, so that NewMadeleine always uses the most appropriate method for the given packet size.
- the *full performance profile* (latency and bandwidth) is stored for a large spectrum of packet size, in order to be able to *predict* the transfer time of a given packet, computed through linear interpolation.

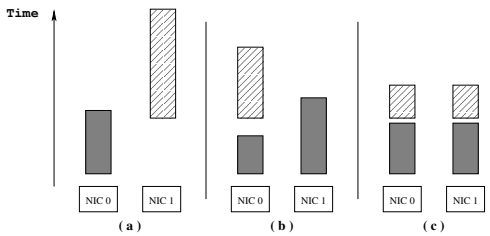

Figure 3. (a) each message is sent on one network. (b) messages are split into chunks of equal size over the available networks. (c) messages are split into chunks of equal transfer time.

B. Auto-tuning thresholds

All thresholds in NEWMADELEINE are auto-tuned through sampling. These thresholds are

- **`iovec`** — The assembly of a message (header+body) may be done by explicit memory copy, or by giving both parts to the network driver in an `iovec`, as explained in Section II-B. We benchmark both methods to determine when it is worth using iovec.
- ***rendezvous*** — We compare the transfer time of both *eager* and *rendezvous* methods to find the *rendezvous* threshold. Since *eager* needs to send chunks scattered in memory, we consider for *eager* the best among `iovec` and memory copy, depending on the threshold determined above. Our *eager* test always does a copy on the receiver side, which is the case when it is unexpected. Moreover, the *rendezvous* threshold is bounded by the maximum size for unexpected messages (64 KB in NEWMADELEINE).
- **aggregation** — We benchmark the time to send two packets and to send an aggregate of the two packets. The aggregation is done either by copy or by `iovec`, depending on the threshold determined above.

All the thresholds are computed by linear interpolation of adjacent points of the crossing.

C. Adaptive load balancing on multi-rail architectures

It is now usual to feature nodes of clusters with several network interface cards. Efficiently exploiting these new rails benefits to most application communication schemes. Smaller data packets can be spread across the available networks, increasing the message rate, while large messages can be split and sent across several links in parallel, increasing the aggregated bandwidth.

In previous papers [3], we already highlight the benefits of splitting large messages and sending the chunks in parallel. Indeed, as it is illustrated in Figure 3(a), sending a message on the first available network interface card might underutilize the communication resources. The major impact would be to only use one rail in the case of applications with sparse communication. Splitting messages in chunks of equal size as illustrated in Figure 3 (b) is only relevant in the case of homogeneous network technologies. With various capabilities, nominal performances of networks can not be reached. Thus, the multi-rail strategy aims at minimizing the transfer time of the whole message by splitting messages in such a way that the time required to send each chunk of a message is equal as the Figure 3 (c) illustrates.

Thanks to the sampling, NEWMADELEINE is able to estimate when a NIC will become idle and to decide which ones may take part to the communication: NICs whose estimated release time is after the whole communication completion by one of the current available interfaces are moved aside. Then, if two NICs have been selected, the split ratio is computed on-the-fly by dichotomy. The algorithm begins by splitting the packets in two chunks of equal size. It then compares the predicted transfer time required by each network for those specific amount of data. In the case of unavailable resources, the required time to finish the current communication is added as a penalty. Chunk size of the fastest network is increased by half of amount of data allocated to the other network. The operation is repeated until both transfer durations are equal.

If there are more than two networks, optimal split ratios determination is more complicated. Our current implementation is based on a weighted average of the measured asymptotic bandwidths. Networks being ordered following their order of participation in the communication, chunk size assigned to the i-th selected network is obtained by summing the amount of data that can be transferred by a NIC before reaching the next point where a network enters in the communication:

$$\sum_{m}^{n}\{\frac{B_i}{\sum_{k=0}^{r} B_k} \cdot data\_size\}$$

where n is the number of time a network joined the communication, m is the step in which the given network takes a part in the communication, B_i is the asymptotic bandwidth of the given network, r the number of networks participating in the current step of communication. $data\_size$ is the total amount of data transferred in the current step:
$data\_size = \sum_{k=0}^{r} B_k \cdot tr_{m+1}$
where tr is the time required to reach the next NIC releasing.

IV. EVALUATION

In this Section, we present NEWMADELEINE sampling results on various systems. We present our experimental testbed comprised of several clusters, equipped with *Myrinet* and *InfiniBand*. Then we present the results and benefits of auto-tuning of transfer method, *rendezvous*, and aggregation thresholds, then we present benchmarks of our adaptive splitting ratio for multi-rail.

A. Experimental testbed

We have conducted our experiments on the following clusters:

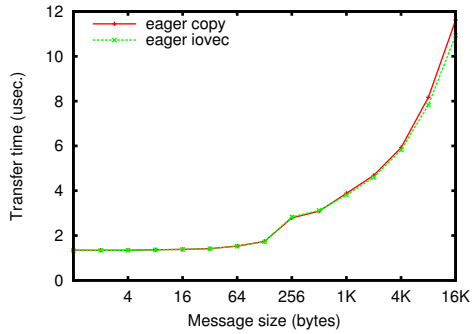

Figure 4. Sampling on *jack*/*InfiniBand*, `iovec` threshold — 8 bytes

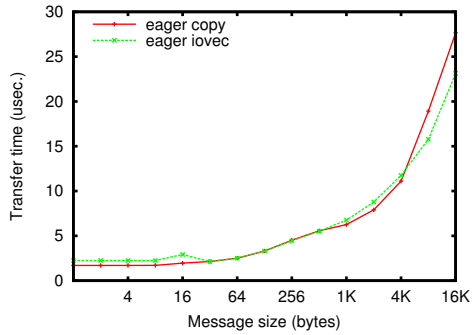

Figure 5. Sampling on *genepi*/*InfiniBand*, `iovec` threshold — 4 KB

- **jack** — Nodes are Intel Xeon X5650, equipped with Mellanox ConnextX2 IB QDR PCIe (MT26428) and Myricom Myri-10G.
- **genepi** — Nodes are Intel Xeon E5420, equipped with Mellanox ConnectX IB DDR PCIe (MT26418).
- **joe** — Nodes are Intel Xeon X5460, equipped with both Mellanox ConnectX IB DDR PCIe (MT25418) and Myricom Myri-10G (10G-PCIE-8A).
- **infini** — Nodes are AMD Opteron 265, equipped with Mellanox InfiniHost III Ex PCI-X (MT25208).

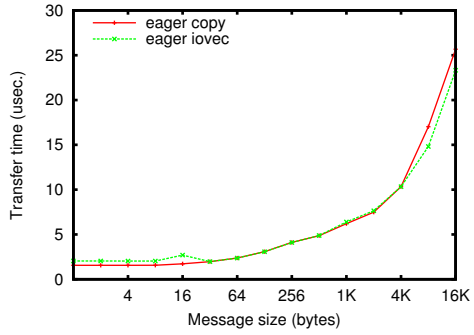

Figure 6. Sampling on *joe*/*InfiniBand*, `iovec` threshold — 49 bytes

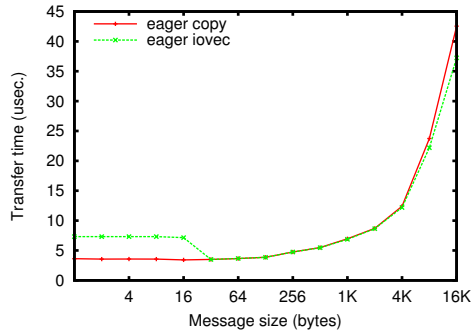

Figure 7. Sampling on *infini*/*InfiniBand*, `iovec` threshold — 96 bytes

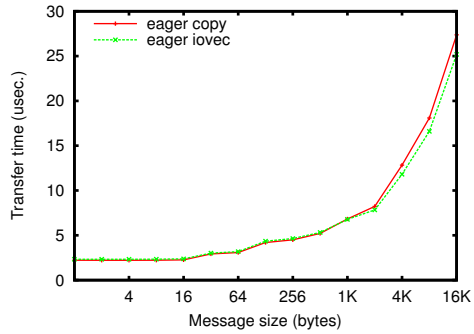

Figure 8. Sampling on *joe*/MX, `iovec` threshold — 868 bytes

B. Transfer methods thresholds tuning

As explained in Section II-B, our NEWMADELEINE communication library builds small packets from a header and a body that have to be assembled into a packet, either through a copy or through an `iovec` directly given to the network driver. Figures 4, 5, 6, 7, and 8 depict the latency of both methods on our various clusters.

We can clearly see that on all of our platforms, the best method for very small messages is to copy, and to use an `iovec` for medium-sized packets. Except for *InfiniBand* being bad for `iovec` below 32 bytes on some platforms (but not all), the differences between these methods may seem small, but it is perceptible with e.g. a difference up to 17 % for 16 KB packets on the *genepi* cluster. We can see that the best threshold, as interpolated by our auto-tuning program, is very different from one cluster to another, from as low as 8 bytes for cluster "jack" to 4 KB for cluster "genepi", even though their configuration may seem very close — both are Xeon with *InfiniBand*.

C. Rendez-vous threshold tuning

Like many communication libraries, NEWMADELEINE can send messages eagerly, at the risk of having a copy on the receiver side if the message is unexpected, or through a *rendezvous* to guarantee that there will be no memory copy. Figures 9, 10, 11, 12, and 13 depict the latency of

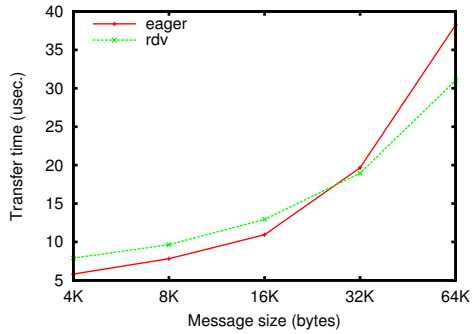

Figure 9. Sampling on *jack*/*InfiniBand*, *rendezvous* threshold — 28 KB

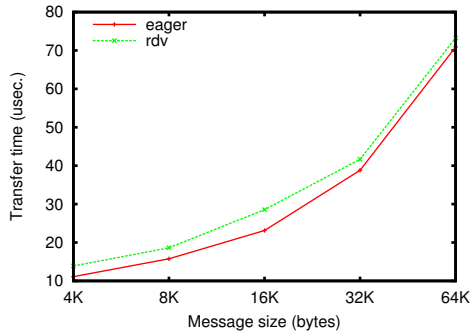

Figure 10. Sampling on *genepi*/*InfiniBand*, *rendezvous* threshold — 64 KB

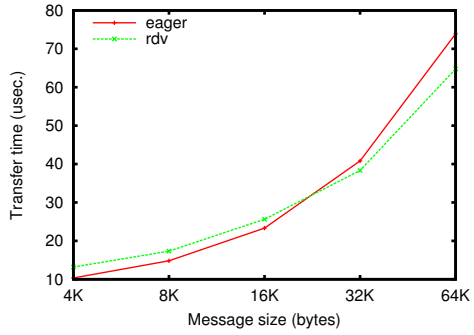

Figure 11. Sampling on *joe*/*InfiniBand*, *rendezvous* threshold — 24 KB

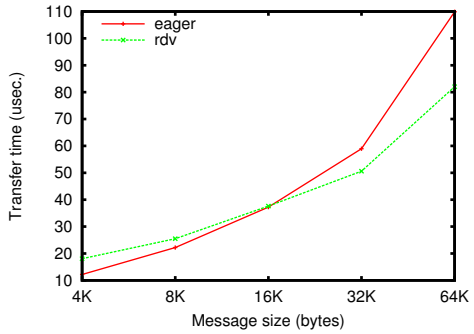

Figure 12. Sampling on *infini*/*InfiniBand*, *rendezvous* threshold — 17 KB

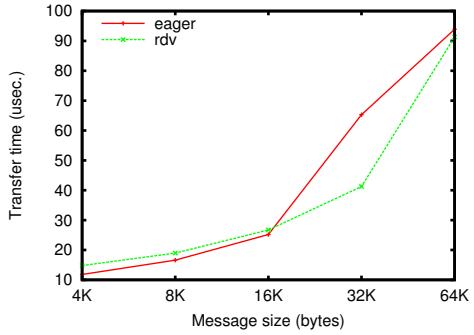

Figure 13. Sampling on *joe*/MX, *rendezvous* threshold — 17 KB

the eager+copy protocol compared to the latency of the *rendezvous* protocol, on our various clusters.

Obviously the best choice is always to send eagerly data for small messages, since the *rendezvous* protocol adds the overhead of a round-trip. However, we can see that there is no "one size fits all" value for this threshold. While a 64 KB threshold (actually the maximum size for unexpected messages) is the best choice on cluster *genepi* (Figure 10), such a hard-coded value would lead to choose the eager mode even for 32 KB messages, which has a latency 84 % higher than *rendezvous* on cluster *joe* (Figure 11). Conversely, choosing a 17 KB hard-coded threshold, which is the best value on cluster *infini* (Figure 12), would lead to a 21 % increase in latency for 17 KB messages on cluster *genepi*.

One may think of choosing the *rendezvous* threshold from the network type — 16 KB for *Myrinet*, larger for *InfiniBand*. However, when comparing results on cluster *genepi* (Figure 10) and on cluster *infini* (Figure 12), both being *InfiniBand*-based, we can see that no such unique value exists for all *InfiniBand*-based clusters, let alone for all networks.

These figures demonstrate that sampling-based auto-tuning for the *rendezvous* threshold brings a significant performance improvement for medium-sized messages over any hard-coded value.

D. Aggregation threshold tuning

As shown in Section II-D, the aggregation-based strategy relies on a threshold to decide whether aggregation is worth or whether it is better to send two separate packets.

Figures 14, 15, 16, 17, and 18 depict the latency of sending two messages as two packets compared to a single packet containing both messages. Aggregation is done using the best method among memory copy or `iovec`, depending on the threshold automatically calculated in previous Section IV-B.

As shown on Figures, not all networks behave equally. Especially on *InfiniBand* clusters, aggregation gives very good results for small messages up to several kilobytes. On

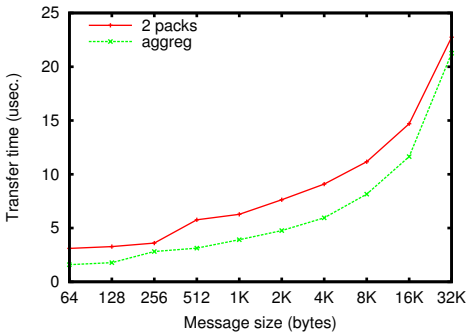

Figure 14. Sampling on *jack*/*InfiniBand*, aggregation threshold — 41 KB.

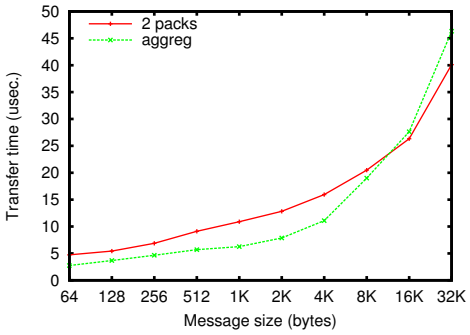

Figure 15. Sampling on *genepi*/*InfiniBand*, aggregation threshold — 12 KB.

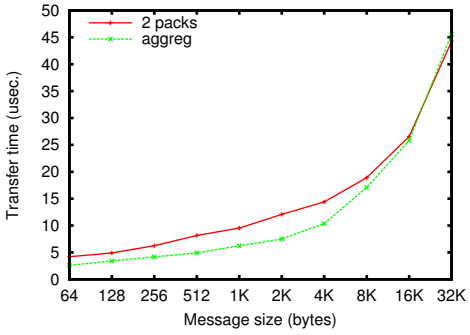

Figure 16. Sampling on *joe*/*InfiniBand*, aggregation threshold — 22 KB.

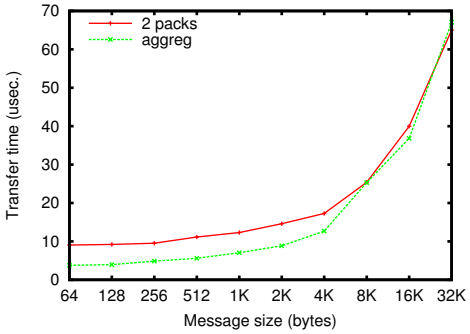

Figure 17. Sampling on *infini*/*InfiniBand*, aggregation threshold — 26 KB.

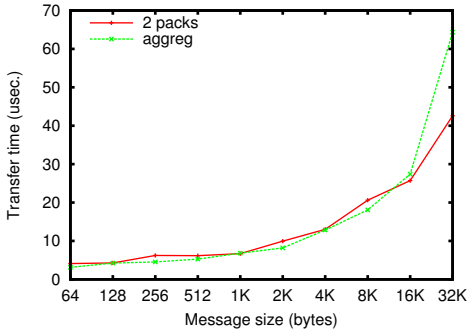

Figure 18. Sampling *joe*/*Myrinet*, aggregation threshold — 950 bytes.

the other hand, the *Myrinet*-based cluster exhibit a not so dramatic advantage of aggregation against two packets, with a quite noticeable performance decrease for aggregation with large messages. This results with an auto-tuning that chooses a large aggregation threshold for *InfiniBand* and quite low for *Myrinet*. It demonstrates that aggregation threshold is different depending on the hardware and that auto-tuning may be performed through sampling to get the right value.

E. Adaptive splitting ratio for multi-rail clusters

As shown in section III-C, the multi-rail strategy relies on the prediction of the networks performances. In order to evaluate the efficiency of this strategy, we have conducted a raw performance evaluation on `joe` and `jack` clusters. We measure the average transfer times with a classical ping-pong test when NEWMADELEINE balances data across MX/MYRINET and VERBS/INFINIBAND. We compare these performance with single rail configurations where NEWMADELEINE only exploits one of these network technologies.

The average round-trip durations on cluster `joe` are reported on Figure 19. While the performance obtained with one network are quite similar before the *rendezvous* threshold, *InfiniBand* performs better for medium-sized messages. The use of both networks simultaneously permits to decrease the transfer time for messages larger than 12 KB. This improvement of the performance is due to the use of two networks to transmit a single message: the data is split into two chunks, each one being transferred through a different NIC. Thus, the transfer time that we measure corresponds to the time required to transmit two small messages in parallel.

Figure 20 shows the bandwidth we measured with this configuration. On both networks, exchanges managed over an exclusive network achieve good performances. Although a light overhead is observed due to NEWMADELEINE internal protocols, bandwidths are close to the nominal ones. We can however see a light discontinuity on *Myrinet* when MX switches internally to a transfer method with a *rendezvous* handshake. The auto-tuning multi-rail feature

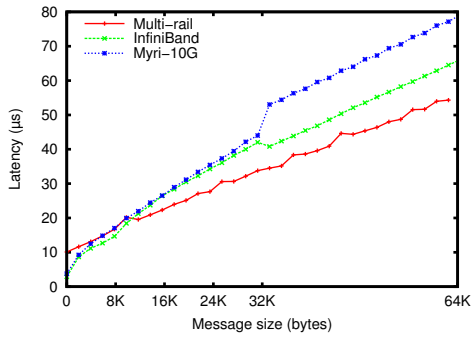

Figure 19. Multi-Rail latency on *InfiniBand* DDR and *Myrinet* on cluster *joe*.

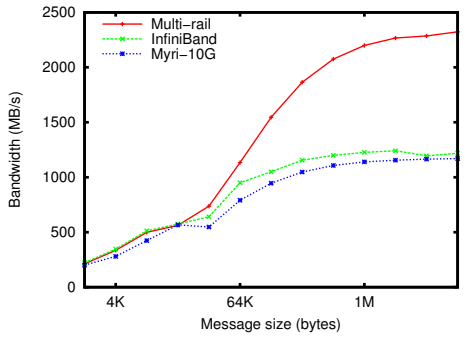

Figure 20. Multi-Rail bandwidth on *InfiniBand* DDR and *Myrinet* on cluster *joe*.

implemented in NEWMADELEINE outperforms the single NIC configurations. This performance improvement is due to NEWMADELEINE ability to split messages and send the resulting chunks over two networks simultaneously. The transfer time is thus reduced and the bandwidth performance reaches 98 % of the nominal aggregate ones.

Figure 21 shows the bandwidth measured on the `jack` cluster for single rail *Myrinet*, *InfiniBand* QDR, *iso-split* multi-rail, and sampling-based multi-rail. Since bandwidths are heterogeneous, iso-split multi-rail performance is low;

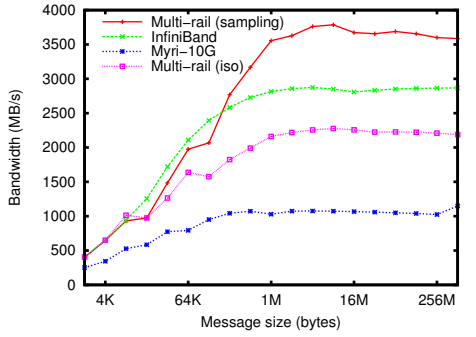

Figure 21. Multi-Rail bandwidth on *InfiniBand* QDR and *Myrinet* on cluster *jack*.

it gets a bandwidth roughly twice the single-rail *Myrinet* bandwidth (as expected), which is lower than single rail *InfiniBand*. On the other hand, sampling-based multi-rail gets a bandwidth roughly equal to the sum of single rail *Myrinet* and *InfiniBand* bandwidth.

This demonstrates that multi-rail is efficient for a wide range of message sizes, depending on the network capabilities. Our sampling-based approach permits to estimate the forthcoming transfer times and thus to adapt the split ratio so that both communication end at the same time, reducing the transmission duration to the minimum.

V. RELATED WORK

Performance auto-tuning is a technique widely-used for computation kernels. It usually consists in observing the application execution to adapt some parameters. BLAS implementations typically rely on auto-tuning techniques. Sampling-based auto-tuning methods are extensively used in the Atlas BLAS implementation [4]. During the installation, Atlas runs a set of benchmarks to determine its optimal parameters such as block size. Another approach to tune BLAS kernels automatically consists in building an historical knowledge base [5]. According to previous executions, the kernel parameters are tuned in order to fit the actual performance of the system. This latter approach is also used in the StarPU runtime system for heterogeneous machines: the performance of computation tasks as well as data transfers between the main memory and the accelerator memory are measured in order to automatically tune the task scheduling [6].

In networking, the AdOC [7] library dynamically adjusts the compression ratio according to the actual relative speed of the CPU and the network, sending extremely reduced packets when the network is busy and submitting raw packets during low activity phases. Gardner *et al.* [8] analyze the congestion of the network at runtime and tune TCP parameters accordingly. OpenMPI provides a tool that runs various benchmarks in order to determine runtime parameters [9]. This work differs from our approach in that it does not permit to predict a transfer duration and thus it cannot balance communication across available NICs.

The problem of load balancing across several parallel network links has been widely studied. Most of these works focus on balancing TCP packets over multiple Ethernet NICs[10], [11], [12]. However, OpenMPI [13] is able to handle several networks and is not bound to Ethernet NICs. By comparing the nominal bandwidth of the available links, OpenMPI splits messages and transmits them across several networks simultaneously. The split may be inaccurate because only asymptotic bandwidth is taken into account, and not the actual value that depends on the message size. A different approach is used in MVAPICH2 [14] for its *InfiniBand* driver: the durations of data transfers are measured on-the-fly and the split ratio may change during the execution

depending on the network congestion. The drawback of this method is that it requires precise measurements in realtime, subject to jitter introduced by applications: applications that overlap communication and computation may delay the detection of a transfer completion, leading to an inaccurate split ratio. These works are different from our approach in that they do not take the NICs workload into account and they only work for homogeneous multi-rail: clusters equipped with several different network technologies are not fully exploited.

VI. Conclusion and future works

In this paper, we have shown that the knowledge of the actual behavior of high performance networks may be used to automatically tune the communication library for best performance. First, the performance reached by a communication library on high performance networks depends on the fine tuning of some parameters, and the best values depend on the actual hardware. We have described and studied a method to automatically adjust these parameters using sampling. Second, we have studied some optimization methods, such as packet aggregation and multi-rail, that greatly benefit from the ability to predict the performance of the hardware to take the best decision automatically. We have described our implementation and evaluated its performance. Our evaluation shows that the best value for some parameters actually vary from one configuration to another, and that our auto-tuning method successfully computes the value that gives the best performance.

This work may be continued in some directions. For now, our sampling does not take the cost of offloading a communication to another core into account. This cost is not negligible and should be taken into account in the performance prediction. However, it is different depending on the pair of cores involved. This question will be investigated carefully in future works.

Adaptive sampling is another direction for future works. Our presented work relies on a sampling pass prior to the application execution. We will investigate the relevance of dynamically sampling the actual communications of the application, without a sampling pass.

Acknowledgments. This work was supported in part by the ANR-JST project FP3C and the ANR project COOP. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

References

[1] H. Subramoni, M. Koop, and D. K. Panda, "Designing Next Generation Clusters: Evaluation of InfiniBand DDR/QDR on Intel Computing Platforms," in *17th Annual Symposium on High-Performance Interconnects (HotI'09)*. IEEE.

[2] O. Aumage, E. Brunet, N. Furmento, and R. Namyst, "NewMadeleine: a Fast Communication Scheduling Engine for High Performance Networks," in *CAC 2007: Workshop on Communication Architecture for Clusters, held in conjunction with IPDPS 2007*. [Online]. Available: http://hal.inria.fr/inria-00127356

[3] E. Brunet, F. Trahay, and A. Denis, "A Multicore-enabled Multirail Communication Engine," in *IEEE International Conference on Cluster Computing*, Tsukuba, Japan, Sep. 2008, pp. 316–321. [Online]. Available: http://hal.inria.fr/inria-00327158

[4] R. C. Whaley and J. Dongarra, "Automatically Tuned Linear Algebra Software," in *Ninth SIAM Conference on Parallel Processing for Scientific Computing*, 1999.

[5] I. Salawdeh, E. César, A. Morajko, T. Margalef, and E. Luque, "Performance model for parallel mathematical libraries based on historical knowledgebase," in *Euro-Par*, 2008.

[6] C. Augonnet, S. Thibault, and R. Namyst, "Automatic Calibration of Performance Models on Heterogeneous Multicore Architectures," in *Proceedings of the International Euro-Par Workshops 2009, HPPC'09*, ser. Lecture Notes in Computer Science. Delft, The Netherlands: Springer, Aug. 2009.

[7] E. Jeannot, "Improving Middleware Performance with AdOC: An Adaptive Online Compression Library for Data Transfer," in *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, 2005.

[8] M. K. Gardner, S. Thulasidasan, and W. chun Feng, "User-Space Auto-Tuning for TCP Flow Control in Computational Grids," *Computer Communications*, vol. 27, 2004.

[9] M. Chaarawi, J. Squyres, E. Gabriel, and S. Feki, "A tool for optimizing runtime parameters of Open MPI," *EuroPVM/MPI*, pp. 210–217, 2008.

[10] T. Shin'ichi Miura, T. Okamoto, and T. Hanawa, "A dynamic routing control system for high-performance PC cluster with multi-path Ethernet connection," in *CAC 2008: Workshop on Communication Architecture for Clusters, held in conjunction with IPDPS 2008*. Miami, FL: IEEE, Apr. 2008.

[11] S. Sumimoto, K. Ooe, K. Kumon, T. Boku, M. Sato, and A. Ukawa, "A scalable communication layer for multi-dimensional hyper crossbar network using multiple gigabit ethernet," in *Proceedings of the 20th annual international conference on Supercomputing*. ACM New York, NY, USA, 2006, pp. 107–115.

[12] S. Karlsson, S. Passas, G. Kotsis, and A. Bilas, "Multiedge: An edge-based communication subsystem for scalable commodity servers," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2007, pp. 1–10.

[13] R. L. Graham, T. S. Woodall, and J. M. Squyres, "Open MPI: A Flexible High Performance MPI," in *The 6th Annual International Conference on Parallel Processing and Applied Mathematics*, 2005.

[14] J. Liu, A. Vishnu, and D. Panda, "Building multirail infiniband clusters: MPI-level design and performance evaluation," in *Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 Conference*, 2004, pp. 33–33.