Short Paper : Dynamic Optimization of Communications over High Speed Networks

Elisabeth Brunet, Olivier Aumage, Raymond Namyst

HAL Id: inria-00110773
https://inria.hal.science/inria-00110773

Submitted on 31 Oct 2006

Short Paper : Dynamic Optimization of Communications over High Speed Networks *

Elisabeth BRUNET   Olivier AUMAGE   Raymond NAMYST

INRIA - LABRI, Université Bordeaux 1,
TALENCE, FRANCE

Abstract

We present a new communication subsystem for high speed networks featuring an extendable packet optimization engine mixing several communication flows. Optimizations are parameterized by the capabilities of the underlying network drivers, and are triggered by the network cards when they become idle. The database of predefined strategies can be easily extended.

1   Introduction

To maintain the performance level of today's high speed networks (such as QUADRICS, MYRINET or IN-FINIBAND), communication libraries combine a variety of techniques and mechanisms to select *how* to send a given packet the *best way*. These techniques include PIO and DMA transfer modes, *eager*, *rendez-vous* and *remote memory access* protocols, zero-copy *vs* by-copy packet handling, etc. Libraries also have to decide whether to aggregate a packet with other packets to reduce the number of network requests –at the cost of additional processing– or to send the packet alone to avoid waiting for another one to merge with (or to benefit from a pipeline effect) or even to use a gather/scatter request. All these decisions must be consistent with the capabilities of the underlying network drivers. The second aspect that makes communication optimization so important is that today's parallel applications tend to use complex conglomerates of multiple communication middlewares such as CORBA, JAVA RMI or DSM, increasing the number of concurrent communication flows between processing nodes. Nowadays communication hardware/software feature NIC virtualization capabilities that provide transparent multiplexing over a single NIC. It would thus be tempting to simply do a one-to-one mapping of communication flows onto virtualized NICS. A much powerful approach, though, is

to consider network multiplexing units as networking resources to be put *in common* into a pool managed by a software packet scheduler. Then, the one-to-one mapping is now only *one* mere scheduling policy (that could be selected as a fallback, for instance) among many other possible ones.

In fact, the role of a modern communication library is to both *optimize and schedule* all the data transfers between processes onto the available network resources.

2   Towards a Dynamic Message Scheduler

Including the application activity as one input parameter of the optimization process and aggregating packets from multiple flows means introducing non-determinism in the underlying communication flows. By having a global control on the network multiplexing resources, a scheduler may assign some of these resources to different classes of traffic (assigning different channel to large synchronous sends, put/get transfers and control/signalling messages) and help the receiver in sorting out the incoming packets. Such a scheduler may also perform dynamic load balancing on multiple resources, multiple NICs, or even NICs from multiple technologies. Finally, the scheduler may also choose to dynamically change the assignment of networking resources to traffic classes, thus selecting different policies, as the needs of the application evolve during the execution.

The MADELEINE library [1] was designed to perform well not only with regular communication schemes –commonly encountered with MPI-like programming environments– but also with programming models involving irregular communication schemes such as RPC or DSM. Other related works [3, 6] have similar optimization features. At the time, though, middleware integrators such as PADICO [2] or code-coupling applications were not widely used: there usually was a single middleware (such as GLOBAL ARRAYS [5], PM2 [4], or a DSM) between the application and the network library. Consequently, this previous version of Madeleine was not designed to perform

cross-flow optimization and its design was limited to deterministic flow manipulations.

3 The Madeleine Optimization Engine

The job of the packet scheduler being to reorganize packets, the more information it is fed with, the better it can optimize the communication flow. One valuable source of information is this communication flow itself: if the scheduler accumulates packets before making a decision, it decreases the risk to take wrong decisions and also widens the possibilities of packet reordering. An example of a wrong decision is to send a small packet just before another small packet becomes available to send, incuring two network transactions where an aggregated one would have been better. For the scheduler to build its own pool of lookahead packets, we propose to synchronize the optimizing process with the NIC(s) activity instead of the application requests. Figure 1 shows this idea: the application simply enqueues packets into a list and immediately returns to computing. The scheduler is not activated each time the application submits a new packet, but rather when one of the NICs becomes idle. While the NIC is busy sending a packet, the scheduler simply accumulates a backlog of packets. If the NIC never stays busy long enough for packets to accumulate, the scheduler may send packets as they become available, as a regular communication library would do, or may artificially delay them for a short time to increase the potential of interesting aggregations (in a TCP NA-GLE's algorithm fashion).

Nevertheless, the very nature of packets encountered in the range of applications targeted by such a communication library must be taken into account. Those applications are built on top of a stack of middlewares. As result, most communication requests will come from these middlewares and will contain extra control data implementing specific protocols. Far from being simple sequences of bytes, such requests are indeed structured messages with one or more fragments expressing *what* the message carries or requests, and one or more other fragments being the actual data or request arguments. These message internal dependencies are expressed by the application and middlewares through the Madeleine API presented in [1]. They are taken into account as limiting factors –or *constraints*– by the scheduler while estimating the value of a given packet reordering operation.

We have implemented this scheduler following the 3-layered architecture depicted on Figure 1. The optimizing layer in the middle monitors NICs activity and makes sure that they are kept *adequately* busy with *ad-*
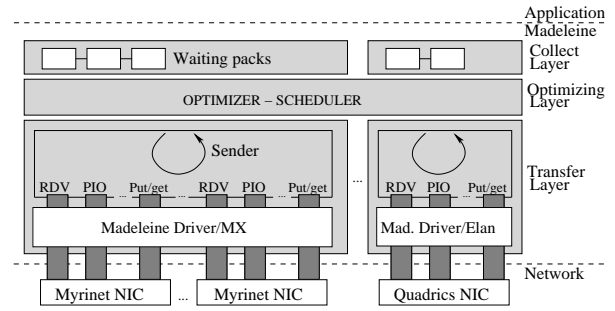

Figure 1. Architecture proposal

equately scheduled communication requests.

4 Conclusion and Future Work

We propose a new approach to optimize communications over high speed networks with a new design of MADELEINE. Its main originality is its powerful and dynamic optimization engine, which is portable, multithreaded, and easily extendable. A beta version on top of MX/Myrinet (available at `http://gforge.inria.fr/projects/pm2`) already exhibits significant improvements over the previous software in many cases. Most noticeably, the aggregation of eager segments collected from several independent communication flows brings huge performance gains.

In the near future, we intend to experiment with different packet lookahead window sizes. We also plan to study how to bound the number of data rearrangements the optimizer has to evaluate so as to determine the best combination of optimization techniques.

References

[1] O. Aumage, L. Bougé, A. Denis, L. Eyraud, J.-F. Méhaut, G. Mercier, R. Namyst, and L. Prylli. A portable and efficient communication library for high-performance cluster computing. *Cluster Computing*, 2002.

[2] A. Denis, C. Pérez, and T. Priol. Padicotm: An open integration framework for communication middleware and runtimes. *Future Generation Computer Systems*, 2003.

[3] W. Gropp, E. Lusk, N. Doss, and A. Skellum. A high performance, portable implementation of the MPI message passing interface standard. Technical Report ANL/MCS-TM-213, Argonne Nat. Lab., 1996.

[4] R. Namyst and J.-F. Méhaut. PM2: Parallel Multi-threaded Machine; A computing environment for distributed architectures. In *Parallel Computing*, 1995.

[5] J. Nieplocha, R. Harrison, and R. Littlefield. Global arrays: a portable "shared-memory" programming model for distributed memory computers. In *SC'94*.

[6] T. Takahashi, S. Sumimoto, A. Hori, H. Harada, and Y. Ishikawa. PM2: High performance communication middleware for heterogeneous network environments. In *SC'00*.