Optimizing Latency in Beowulf Clusters

Rafael Garabato[1], Andrés More[1 2], and Victor Rosales[1]

[1] Argentina Software Design Center (ASDC - Intel Córdoba)
[2] Instituto Universitario Aeronáutico (IUA)

Abstract. This paper discusses how to decrease and stabilize network latency in a Beowulf system. Having low latency is particularly important to reduce execution time of High Performance Computing applications. Optimization opportunities are identified and analyzed over the different system components that are integrated in compute nodes, including device drivers, operating system services and kernel parameters.

This work contributes with a systematic approach to optimize communication latency, provided with a detailed checklist and procedure. Performance impacts are shown through the figures of benchmarks and mpiBLAST as a real-world application. We found that after applying different techniques the default Gigabit Ethernet latency can be reduced from about 50 μs into nearly 20 μs.

Keywords: Beowulf, Cluster, Ethernet, Latency

1 Introduction

1.1 Beowulf Clusters

Instead of purchasing an expensive and high-end symmetric multiprocessing (SMP) system, most scientists today choose to interconnect multiple regular-size commodity systems as a means to scale computing performance and gain the ability to resolve bigger problems without requiring heavy investments [14] [13] [16].

The key driving factor is cost, hence out-of-the-box hardware components are used together with open source software to build those systems. In the specific case of academia, open source software provides the possibility to make software stack modifications, therefore enabling innovation and broadening their adoption.

Clusters are nearly ubiquitous at the Top500 ranking listing most powerful computer systems worldwide, clustered systems represent more than 80% of the list (Figure 1).

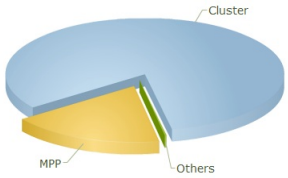

Fig. 1. Top500 System Share by Architecture (as of June 2012)

As the cheapest network fabrics are the ones being distributed on-board by system manufacturers, Ethernet is the preferred communication network in Beowulf clusters. At the moment Gigabit Ethernet is included integrated on most hardware.

1.2 Latency

Latency itself can be measured at different levels, in particular communication latency is a performance metric representing the time it takes for information to flow from one compute node into another. It then becomes not only important to understand how to measure the latency of the cluster but also to understand how this latency affects the performance of High Performance applications [12].

In the case of latency-sensitive applications, messaging needs to be highly optimized and even be executed over special-purpose hardware. For instance latency directly affects the synchronization speed of concurrent jobs in distributed applications, impacting their total execution time.

1.3 Related Work

There are extensive work on how to reduce communication latency [10] [6]. However, this work contributes not with a single component but with a system wide point of view.

The top supercomputers in the world report latencies that commodity systems cannot achieve (Table 1). They utilize specially built network hardware, where the cost factor is increased to get lower latency.

Table 1. Communication Latency at the HPCC ranking

System	**Latency**	**Description**
HP BL280cG65	0.49 μsec	Best Latency
Fujitsu K Computer	6.69 μsec	Top System

High performance network technology (like InfiniBand [2]) is used in cases were state-of-the-art Ethernet cannot meet the required latency (see reference

values in Table 2). Some proprietary network fabrics are built together with supercomputers when they are designed from scratch.

Table 2. System Level Ethernet Latency

Latency	Technology
30-125 μsec	1Gb Ethernet
5-30 μsec	10Gb Ethernet

1.4 Problem Statement

The time it takes to transmit on a network can be calculated as the required time a message information is assembled and dissembled plus the time needed to transmit message payload. Equation 1 shows the relation between these startup plus throughput components for the transmission of n bytes.

$$t(n) = \alpha + \beta \times n \tag{1}$$

In the hypothetical case where *zero bytes* are transmitted, we can get the minimum possible latency on the system (Equation 2). The value of α is also known as the theoretical or zero-bytes latency.

$$t(0) = \alpha \tag{2}$$

It is worth noticing that α is not the only player in the equation, $1/\beta$ is called network bandwidth, the maximum transfer rate that can be achieved. β is the component that affects the overall time as a function of the package size.

2 Benchmarking Latency

There are different benchmarks used to measure communication latency.

2.1 Intel MPI Benchmarks

The Intel MPI Benchmarks (IMB) are a set of timing utilities targeting most important Message Passing Interface (MPI) [7] functions. The suite covers the different versions of the MPI standard, and the most used utility is Ping Pong.

IMB Ping Pong performs a single message transfer exercise between two active MPI processes (Figure 2). The action can be run multiple times using varying message lengths, timings are averaged to avoid measurement errors.

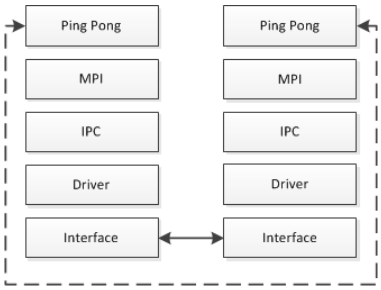

Fig. 2. IMB Ping Pong Communication

Using only MPI basic routines, a package is sent (`MPI_SEND`) from a host system and received (`MPI_RECV`) on a remote one (Figure 3) and the time is reported as half the time in μs for an X long bytes (`MPI_BYTE`) package to complete a round trip.

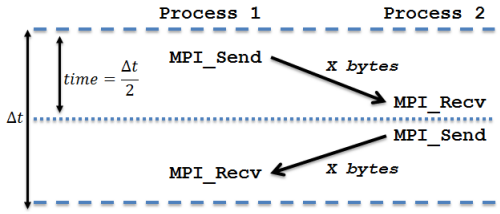

Fig. 3. IMB Ping Pong Benchmark

As described by the time formula at Equation 1, different measures of transmission time are obtained depending on the package size. To get the minimum latency an empty package is used.

2.2 Other Benchmarks

There are other relevant HPC benchmarks that are usually used to exercise clusters: HPL and HPCC. These exercise the system from an application level, integrating all components performance for a common goal.

It is worth mentioning that there are other methods that work at a lower level of abstraction, for instance using Netperf [9] or by following RFC 2544 [3] techniques. However these last two measure latency at network protocol and device level respectively.

High Performance Linpack High Performance Linpack is a portable benchmark for distributed-memory systems doing pure matrix multiplication [1]. It

provides a testing and timing tool to quantify cluster performance. It requires MPI and BLAS supporting libraries.

High Performance Computing Challenge Benchmarks The HPC Challenge benchmark suite [5] packages 7 benchmarks:

HPL: measures floating point by computing a system of linear equations.
DGEMM: measures the floating point rate of execution of double precision real matrix-matrix multiplication.
STREAM: measures sustainable memory bandwidth.
PTRANS: computes a distributed parallel matrix transpose
RandomAccess: measures random updates of shared distributed memory
FFT: double precision complex one-dimensional discrete Fourier transform.
b_eff: measures both communication latency and bandwidth

HPL, DGEMM, STREAM, FFT run in parallel in all nodes, so they can be used to check if cluster nodes are performing similarly. PTRANS, RandomAccess and b_eff exercise the system cluster wide. It is expected that latency optimizations impact their results differently.

3 Methods

Given a simplified system view of a cluster, there are multiple compute nodes that together run the application. An application uses software such as libraries that interface with the operating system to reach hardware resources through device drivers. This work analyzes the following components:

Ethernet Drivers: interrupt moderation capabilities
System Services: interrupt balancing and packet-based firewall
Kernel Settings: low latency extensions on network protocols

Further work to optimize performance is always possible; only the most relevant optimizations were considered according to gathered experience over more than 5 years on the engineering of volume HPC solutions.

3.1 Drivers

As any other piece of software, device drivers implement algorithms which, depending on different factors, may introduce latency. Drivers may even expose hardware functionalities or configurations that could change the device latency to better support the Beowulf usage scenario.

Interrupt Moderation Interrupt moderation is a technique to reduce CPU interrupts by caching them and servicing multiple ones at once [4]. Although it make sense for general purpose systems, this introduces extra latency, so Ethernet drivers should not moderate interruptions when running in HPC clusters.

To turn off Interrupt Moderation on Intel network drivers add the following line on each node of the cluster and reload the network driver kernel module. Refer to documentation [8] for more details.

```
# echo "options e1000e InterruptThrottleRate=0" > /etc/modprobe.conf
# modprobe -r e1000e && modprobe e1000e
```

For maintenance reasons some Linux distributions do not include the configuration capability detailed above. In those cases, the following command can be used to get the same results.

```
# ethtool eth0 rx-usecs
```

There is no portable approach to query kernel modules configurations in all Linux kernel versions, so configuration files should be used as a reference.

3.2 Services

Interrupt Balancing Some system services may directly affect network latency. For instance *irqbalance* job is to distribute interrupt requests (IRQs) among processors (and even between each processor cores) on a *Symmetric Multi-Processing* (SMP) system. Migrating IRQs to be served from one CPU to another is a time consuming task that although balance the load it may affect overall latency.

The main objective of having such a service is to balance between power-savings and optimal performance. The task it performs is to dynamically distribute workload evenly across CPUs and their computing cores. The job is done by properly configuring the IO-ACPI chipset that maps interruptions to cores.

An ideal setup will assign all interrupts to the cores of a same CPU, also assigning storage and network interrupts to cores near the same cache domain. However this implies processing and routing the interrupts before running them, which has the consequence of adding a short delay on their processing.

Turning off the *irqbalance* service will help then to decrease network latency. In a Red Hat compatible system this can be done as follows:

```
# service irqbalance stop
# chkconfig irqbalance off
$ service irqbalance status
```

Firewall As compute nodes are generally isolated on a private network reachable only through the head node, the firewall may not even be required. The system firewall needs to review each package received before continuing with the execution. This overhead increases the latency as incoming and outgoing packet fields are inspected during communication.

Linux-based systems have a firewall in its kernel that can be controlled throughout a user-space application called *iptables*. This application runs in the system as a service, therefore the system's service mechanisms has to be used to stop it.

```
# service iptables stop
# chkconfig iptables stop
$ lsmod | grep iptables
```

3.3 Kernel Parameters

The Linux Transport Control Protocol (TCP) stack makes decisions by default that favors higher throughput as opposed to low latency. The Linux TCP stack implementation has different packet lists to handle incoming data, the PreQueue can be disabled so network packets will go directly into the Receive queue. In Red Hat compatible systems this can be done with the command:

```
# echo 1 > /proc/sys/net/ipv4/tcp_low_latency
$ sysctl -a | grep tcp_low_latency
```

There are others parameters that can be analyzed [15], but the impact they cause are too application specific to be included on a general optimization study.

4 Optimization Impact

4.1 IMB Ping Pong

Using IMB Ping Pong as workload, the following results (Figure 4) reflect how the different optimizations impact communication latency. The actual figures on average and deviation are shown below at Table 3.

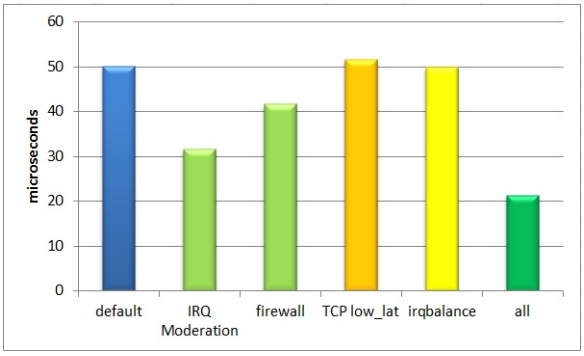

Fig. 4. Comparison of Optimizations

Table 3. IMB Ping Pong Optimization Results

Optimization	$\tilde{\mathbf{x}}$ (σ^2)	**Impact**
Default	50.03 (4.31)	N/A
IRQ Moderation	31.63 (0.83)	36.79%
Firewall	41.62 (8.90)	16.82 %
TCP LL	51.59 (8.22)	-3.11%
IRQ Balance	49.72 (9.68)	0.62 %
Combined	21.31 (2.09)	57.40 %

The principal cause of overhead in communication latency is then IRQ moderation. Another important contributor is the packet firewall service. We found that the low latency extension for TCP was actually slightly increasing the IMB Ping Pong reported latency. In the case of the IRQ balance service, the impact is only minimal.

Optimizations impact vary, and not surprisingly they are not accumulative when combining them all. At a glance, it is possible to optimize the average latency in nearly 54%, nearly halving result deviations.

4.2 High Performance Linpack

A cluster-wide HPL running over MPI reported results as shown in Table 4. The problem size was customized to `Ns:37326 NBs:168 Ps:15 Qs:16` for a quick but still representative execution with a controlled deviation.

Table 4. HPL Results

Optimization	Wall-time	Gflops
Default	00:20:46	0.02921
Optimized	00:09:03	0.07216

As we can see on the results, the actual synchronization cycle done by the algorithm heavily relies on having low latency. The linear system is partitioned in smaller problem blocks which are distributed over a grid of processes which may be on different compute nodes. The distribution of matrix pieces is done using a binary tree among compute nodes with several rolling phases between them. The required time was then reduced 56%, and the gathered performance was increased almost 2.5 times.

4.3 HPCC

Figure 5 and table 5 show HPCC results obtained with a default and optimized Beowulf cluster. As we can see on the results, the overall execution time is directly affected with a 29% reduction. The performance figures differ across packaged benchmarks as they measure system characteristics that are affected by latency in diverse ways.

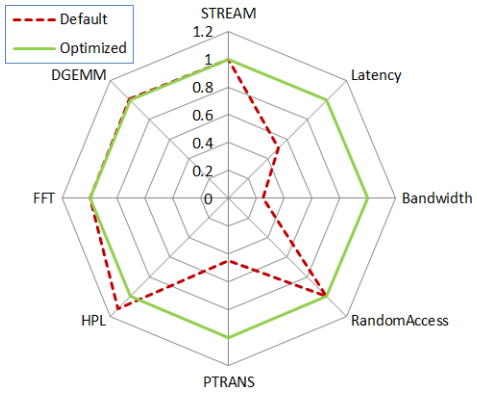

Fig. 5. HPCC Performance Results (higher is better)

Table 5. HPCC Timing Results

Optimization	Wall-time
Default	00:10:32
Optimized	00:07:27

Local benchmarks like STREAM, DGEMM and HPL are not greatly affected, as they obviously do not need communication between compute nodes. However, the actual latency, bandwidth and PTRANS benchmark are impacted as expected due they communication dependency.

4.4 mpiBLAST

In order to double check if any of the optimization have hidden side effects and the real impact on the execution of a full-fledge HPC application, a real-world code was exercised. mpiBLAST [11] is an open source tool that implements DNA-related algorithms to find regions of similarity between biological sequences.

Table 6 shows the actual averaged figures after multiple runs. Results got with a default and optimized system on a fixed workload for mpiBLAST. The required time to process the problem was reduced by 11% with the previous 42% improvement as measured by IMB Ping Pong.

Table 6. mpiBLAST Results

Optimization	**Wall-time**
Default	534.33 seconds
Optimized	475.00 seconds

This shows that the results of a synthetic benchmark like IMB Ping Pong can not be used directly to extrapolate figures, they are virtually the limit to what can be achieved by an actual application.

4.5 Testbed

The experiments done as part of this work were done over 32 nodes with the following bill of materials (Table 7).

Table 7. Compute Node Hardware and Software

Component	**Description**
Server Board	Intel(R) S5000PAL
CPU	Intel(R) Xeon(R) X5355 @ 2.66GHz
Ethernet controller	Intel(R) 80003ES2LAN (Copper) (rev 01)
RAM Memory	4 GB DDR2 FB 667 MHz
Operating System	Red Hat Enterprise 5.5 (Tikanga)
Network Driver	Intel(R) PRO/1000 1.2.20-NAPI
Ethernet Switch	Hewlett Packard HPJ4904A

5 Optimization Procedure

Figure 6 summarizes the complete optimization procedure. It is basically a sequence of steps involving checking and reconfiguring Ethernet drivers and system services if required. Enabling TCP extensions for low latency is not included due their negative consequences.

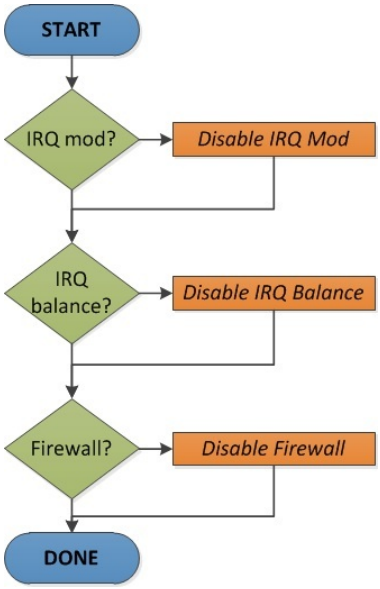

Fig. 6. Latency Optimization Procedure

5.1 Detailed Steps

The steps below include the purpose and an example of the actual command to execute as required on Red Hat compatible systems. The pdsh[3] parallel shell is used to reach compute nodes at once.

Questions (1) helps to dimension the required work to optimize driver configuration to properly support network devices. Questions (2) helps to understand what's needed to properly configure system services.

1. Interrupt Moderation on Ethernet Driver
   (a) Is the installed driver version the latest and greatest?

```
$ /sbin/modinfo -F version e1000e
1.2.20-NAPI
```

[3] `http://sourceforge.net/projects/pdsh`

(b) Is the same version installed across all compute nodes?

```
$ pdsh -N -a '/sbin/modinfo -F version e1000e' | uniq
1.2.20-NAPI
```

(c) Are interrupt moderation settings in HPC mode?

```
# pdsh -N -a 'grep "e1000e" /etc/modprobe.conf' | uniq
options e1000e InterruptThrottleRate=0
```

2. System Services

(a) Is the firewall disabled?

```
# pdsh -N -a 'service iptables status' | uniq
Firewall is stopped.
```

(b) Is the firewall disabled at startup?

```
# pdsh -N -a 'chkconfig iptables --list'
irqbalance 0:off 1:off 2:off 3:off 4:off 5:off 6:off
```

(c) Was the system rebooted after stopping firewall services?

```
$ uptime
 15:42:29 up 18:49, 4 users, load average: 0.09, 0.08, 0.09
```

(d) Is the IRQ balancing service disabled?

```
# pdsh -N -a 'service irqbalance status' | uniq
irqbalance is stopped
```

(e) Is IRQ balancing daemon disabled at startup?

```
# pdsh -N -a 'chkconfig irqbalance --list' | uniq
irqbalance 0:off 1:off 2:off 3:off 4:off 5:off 6:off
```

Once gathered all the information required to known if optimizations can be applied, the following list can be used to apply configuration changes. Between each change a complete cycle of measurement should be done. This include contrasting old and new latency average plus their deviation using at least IMB Ping Pong.

Disable IRQ Moderation

```
# pdsh -a 'echo "options e1000e InterruptThrottleRate=0" >> \
/etc/modprobe.conf'
# modprobe -r e1000e; modprobe e1000e
```

Disable IRQ Balancer

```
# pdsh -a 'service irqbalance stop'
# pdsh -a 'chkconfig irqbalance off'
```

Disable Firewall

```
# pdsh -a 'service iptables stop'
# pdsh -a 'chkconfig iptables off'
```

6 Conclusion

This work shows that by only changing default configurations the latency of a Beowulf system can be easily optimized, directly affecting the execution time of High Performance Computing applications. As a quick reference, an out-of-the-box system using Gigabit Ethernet has around 50 μs of communication latency. Using different techniques, it is possible to get as low as nearly 20 μs.

After introducing some background theory and supporting tools, this work analyzed and exercised different methods to measure latency (IMB, HPL and HPCC benchmarks). This work also contrasted those methods and provided insights on how they should be executed and their results analyzed.

We identified which specific items have higher impact over latency metrics (interrupt moderation and system services), using de-facto benchmarks and a real-world application such as mpiBLAST.

6.1 Future Work

Running a wider range of real-world computational problems will help to understand the impact in different workloads. A characterization of the impact according to the application domain, profiling information or computational kernel might be useful to offer as a reference.

There are virtually endless opportunities to continue with the research on latency optimization opportunities; among them components like BIOS, firmware, networking switches and routers. An interesting opportunity are the RX/TX parameters of Ethernet drivers that control the quantity of packet descriptors used during communication.

Another option is to implement an MPI trace analysis tool to estimate the impact of having an optimized low latency environment. At the moment there are several tools to depict communication traces (Jumpshot[4], Intel's ITAC[5]), but they do not provide a simulation of what would happen while running over a different network environment. Having this approximation can be useful to decide if it is worth to purchase specialized hardware or not.

At last, it would be interesting also to understand the impact of this work into research or development processes using clusters, not only in industry but also in academia.

Acknowledgments

The authors would like to thanks the Argentina Cluster Engineering team at the Argentina Software Design Center (ASDC Intel) for their contributions.

[4] `http://www.mcs.anl.gov/research/projects/perfvis/software/viewers`
[5] `http://software.intel.com/en-us/articles/intel-trace-analyzer`

References

1. J. Dongarra A. Petitet, R. C. Whaley and A. Cleary. A portable implementation of the high-performance linpack benchmark for distributed-memory computers. Technical report, 2008.
2. Infiniband Trade Association. Infiniband architecture specification release 1.2.1. Technical report, 2008.
3. S. Bradner and J. McQuaid. Ieee rfc2544: Benchmarking methodology for network interconnect devices, 1999.
4. Intel Corporation. Interrupt Moderation Using Intel Gigabit Ethernet Controllers Application Note . Technical report, 2007.
5. Jack J. Dongarra, I. High, and Productivity Computing Systems. Overview of the hpc challenge benchmark suite, 2006.
6. A.P. Foong, T.R. Huff, H.H. Hum, J.R. Patwardhan, and G.J. Regnier. Tcp performance re-visited. In *Performance Analysis of Systems and Software, 2003. ISPASS. 2003 IEEE International Symposium on*, pages 70 – 79, march 2003.
7. Message Passing Interface Forum. Mpi: A message-passing interface standard. Technical report, 2009.
8. Improving Measured Latency in Linux for Intel(R) 82575/82576 or 82598/82599 Ethernet Controllers. Interrupt moderation using intel gigabit ethernet controllers application note. Technical report, 2009.
9. R. Jones. Netperf, 2007.
10. S. Larsen, P. Sarangam, and R. Huggahalli. Architectural breakdown of end-to-end latency in a tcp/ip network. In *Computer Architecture and High Performance Computing, 2007. SBAC-PAD 2007. 19th International Symposium on*, pages 195 –202, oct. 2007.
11. Heshan Lin, Pavan Balaji, Ruth Poole, Carlos Sosa, Xiaosong Ma, and Wu-chun Feng. Massively parallel genomic sequence search on the blue gene/p architecture. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 33:1–33:11, Piscataway, NJ, USA, 2008. IEEE Press.
12. QLogic. Introduction to ethernet latency. Technical report, 2011.
13. John Salmon, Christopher Stein, and Thomas Sterling. Scaling of beowulf-class distributed systems. In *In Proceedings of SC'98*, 1998.
14. Thomas Sterling, Donald J. Becker, Daniel Savarese, John E. Dorband, Udaya A. Ranawake, and Charles V. Packer. Beowulf: A parallel workstation for scientific computation. In *In Proceedings of the 24th International Conference on Parallel Processing*, pages 11–14. CRC Press, 1995.
15. Assigning Interrupts to Processor Cores using an Intel(R) 82575/82576 or 82598/82599 Ethernet Controller. Interrupt moderation using intel gigabit ethernet controllers application note. Technical report, 2009.
16. Ewing Lusk William Gropp and Thomas Sterling. *Beowulf Cluster Computing with Linux, Second Edition*. The MIT Press, 2003.