Comparing the Communication Performance and Scalability of a Linux and a NT Cluster of PCs, a Cray Origin 2000, an IBM SP and a Cray T3E-600

Glenn R. Luecke, Bruno Raffin and James J. Coyle

Iowa Sate University
Ames, Iowa 50011-2251 USA
grl@iastate.edu, raffin@iastate.edu, jjc@iastate.edu

Abstract

This paper presents scalability and communication performance results for a cluster of PCs running Linux with the GM communication library, a cluster of PCs running Windows NT with the HPVM communication library, a Cray T3E-600, an IBM SP and a Cray Origin 2000. Both PC clusters were using a Myrinet network. Six communication tests using MPI routines were run for a variety of message sizes and numbers of processors. The tests were chosen to represent commonly-used communication patterns with low contention (a ping-pong between processors, a right shift, a binary tree broadcast and a synchronization barrier) to communication patterns with high contention (a naive broadcast and an all-to-all).

For most of the tests the T3E provides the best performance and scalability. For an 8 byte message the NT cluster performs about the same as the T3E for most of the tests. For all the tests but one, the T3E, the Origin and the SP outperform the two clusters for the largest message size (10 Kbytes or 1 Mbyte).

1 Introduction

In the last several years, much effort has gone into developing low cost, high-performance parallel machines built with PCs [9]. This paper compares the communication performance and scalability of a cluster of PCs running Linux with the GM communication library, a cluster of PCs running Windows NT with the HPVM communication library, a Cray T3E-600, an IBM SP and a Cray Origin 2000. Both PC clusters were using a Myrinet network. Six communication tests that implement commonly-used communication patterns with low contention (a ping-pong between processors, a right shift, a binary tree broadcast and a synchronization barrier) to communication patterns with high contention (a naive broadcast and an all-to-all) were run on these machines.

All tests were written in Fortran 90 and use the `mpi_barrier`, `mpi_send`, `mpi_recv` and `mpi_sendrecv` MPI communication routines. The MPI communication library [15] was chosen because MPI was the only method of performing communication available on all the machines used. Except for the synchronization barrier test, we did not use any MPI collective communication routines but used our own implementations in order to guarantee that the same algorithm is being used for each test on each machine. For example, if `mpi_bcast` had been used to evaluate the performance of the broadcast operation, one machine may use a binary tree algorithm for all message sizes to implement `mpi_bcast` whereas another machine may use different algorithms depending on the message size [8] and the number of processors being used. This would make it difficult to compare the communication performance of these machines.

The Linux cluster of PCs [1] used was a 128 processor machine located at the Albuquerque High Performance Computing Center in Albuquerque, New Mexico. This cluster was a 64-node AltaCluster, by Alta Technology Corporation, each node consisting of a dual processor Intel 450 MHz Pentium II. The NT cluster of PCs [5] used was a 128 processor machine located at the National Center for Supercomputing Applications in Urbana-Champaign, Illinois. This cluster was a 64-node machine, each node consisting of a dual processor Intel 550 MHz Xeon Pentium III. The Origin 2000 [2, 6, 10] used was a 128 processor machine located in Eagan, Minnesota. This Origin was a 64-node machine, each node consisting of two 300 MHz MIPS R12000 processors sharing a common memory. The T3E-600 [2, 14, 7] used was a 512 processor machine located in Eagan, Minnesota. Each processor was a DEC Alpha EV5 microprocessor running at 300 MHz. The IBM SP [3, 4] used was a 251 processor machine located at the Maui High Performance Computing Center, Hawaii. Because of limited access to the machine, a maximum of 96 processors (P2SC microprocessors running at 160 MHz) were available for running our tests. For more information about these machines and the software used, please refer to the expanded version of this paper which can be found at [12].

Tests were run with 16, 32, 48, 64 processors for the Linux cluster, 16, 32, 48, ...,96 processors for the IBM SP and 16, 32, 48, ..., 128 processors for all the other machines. More processors were available on the T3E-600, but for sake of clarity we do not present results for more than 128 processors. Tests run on the Origin and the T3E-600 were executed on machines dedicated to running only these tests. Tests run on the IBM SP, the Linux cluster of PCs and the NT cluster of PCs were executed using IBM's scheduler Loadleveler, PBS scheduler and LSF scheduler, respectively, so that only one job at a time would be executing on the requested processors. Tests were run using the default setting for each machine. All messages used 8 byte reals. The tests were run with messages of size 8 bytes and 10 Kbytes for the all-to-all and the naive broadcast tests, and with 8 bytes, 10 Kbytes and 1 Mbyte for the other tests. Throughout this paper 1 Kbyte means 10^3 bytes and 1 Mbyte means 10^6 bytes. Limited access to the machines and memory limitations prevented us from running these tests with more or larger message sizes and for all possible number of processors. All tests could have been run with a job requesting the maximum number of processors available even though a test might only require 2 processors. When this same test was run requesting only 2 processors instead of the maximum number of processors available, the measured performance would usually increase. Thus, tests requiring p processors were run with exactly p processors requested.

2 Timing Methodology

All the tests have been timed using the code listed below:

```
parameter (ntest=51)
real*8, dimension(ntest) ::  time
real*8, dimension(ntest) ::  pe_time
...
do k = 1, ntest
   cache = cache + 0.1
   call mpi_barrier(mpi_comm_world,ierr)
   time1 = mpi_wtime()
   ...  MPI code to be timed ...
   time2 = mpi_wtime()
   pe_time(k) = time2 - time1
   call mpi_barrier(mpi_comm_world,ierr)
end do
call mpi_reduce(pe_time,time,ntest,&
     mpi_real8,mpi_max,0,&
     mpi_comm_world,ierr)
```

Timings were done by first flushing the cache on all processors by changing the values in the real array `cache(1:ncache)`, prior to timing the desired operation. The value `ncache` was chosen so the size of `cache` was the size of the secondary cache for the T3E (96*1024 bytes), the Origin 2000 (8*1024*1024 bytes) and the 2 clusters (512*1024 bytes), and the size of the primary cache for the IBM SP (64*1024 bytes) since this machine does not have a secondary cache. Note that by flushing the cache between each trial, the data that may have been loaded in the cache during the previous trial cannot be used to optimize the communications of the next trial, which a machine like the Origin 2000 could do [6].

The first call to `mpi_barrier` guarantees that all processors reach this point before they each call the wall-clock timer, `mpi_wtime`. The second call to a synchronization barrier is to ensure that no processor starts the next iteration (flushing the cache) until all processors have completed executing the "MPI code to be timed". The test is executed `ntest` times and the values of the differences in times on each participating processor are stored in `pe_time`. The call to `mpi_reduce` calculates the maximum of `pe_time(k)` over all participating processors for each fixed `k` and places this maximum in `time(k)` for all values of `k`. Thus, `time(k)` is the time to execute the test for the k-th trial.

The resolution of the timer routines `mpi_wtime()` is 0.013 microseconds on the T3E and the SP, 0.800 microseconds on the Origin and 0.953 microseconds on the Linux cluster and 1 microsecond on the NT cluster. This is not accurate enough to time the MPI synchronization barrier test. Thus, for this test we timed the "code to be timed" 20 times between the two calls to `mpi_wtime()`. Of course, the values in the array `time(1:ntest)` are all divided by 20 so they reflect the time for a single iteration of the test.

Observe from the above timing code that one run will measure the execution time of the "code to be timed" `ntest` times. For all the tests on the T3E, the SP and the clusters, the measured first time was usually significantly larger than most of the subsequent times. Thus, for these tests `ntest` was set to 51 and the first measured time was thrown away. Examination of data showed that there was no need to take larger values of `ntest`. The remaining data was then filtered as described below.

Timing for the MPI tests on the Origin was more challenging. Figure 1 page 5 shows execution times for the all-to-all test with `ntest` = 100. About the first 20 timings showed significant variation and then the timings would settle down. For this reason, for all the MPI tests executed on the Origin, `ntest` was set to 100 and only the last 50 timings were used. The remaining data was filtered as described below. This behavior was not observed on the other machines.

All tests were run twice making 100 remaining trials for each test for a given message size and for a given number of processors. Among these trials, sometimes a few would be significantly larger than the other and would make the average over all trials to significantly increase (see [11] for more details about this behavior). These time "spikes" are likely due to operating system interruptions. It was our opinion that these spikes should be removed so that the performance data will reflect the times one will usually obtain. All data in this report has the spikes removed by the following process: First the median is computed and all times greater than 1.8 times this median are removed. There were a few cases where the above procedure would remove more than 10% of the data. We felt that this would not be appropriate, so in these cases only the 10% of the largest times were removed. An average was then calculated from the filtered data and this is what is presented for each test in this report. For all tests and all message sizes, this filtering process led to the removal of 0.57% of the trials for the Origin, 0.45% for the T3E, 3.64% for the SP, 1.61% for the NT cluster and 0.71% for the Linux cluster.

3 Communication Tests and Performance Results

Due to page limitations the codes used for the tests are not presented in this paper but can be found in [12].

Test 1: The Synchronization Barrier

This test evaluates the time to execute the MPI synchronization barrier, `mpi_barrier`. Figure 2, page 5, presents the performance data for this test. The T3E and the Origin scale and perform significantly better than the SP and the two clusters. Also notice that the NT cluster outperforms the SP and the Linux cluster.

Test 2: Ping-Pong Between Processors

The purpose of this test is to determine the performance differences for sending messages between "close" processors and "distant" processors. In a perfectly scalable computer the time required to send a message would be independent of the processors used. This test measures the time required for processor 0 to send a message and receive it back from processor j for $j = 1$ to 96 for the SP and for $j = 1$ to 128 for the other machines. Because of time limitations we did not test ping-pong times between all processors.

The results of this test are based on one run per machine because the assignment of ranks to physical processors will vary from one run to another. Figure 3, page 6, presents the performance data for this test. Each reported time is divided by two. For all 3 message sizes, the T3E shows the best performance. In a perfectly scalable computer, these graphs would all be horizontal lines. Notice that many of these graphs are "reasonably close" to this ideal.

Test 3: The Right Shift

For the right shift test each processor gets a message of size `n` from its left neighbor. In a perfectly scalable computer the execution time for this test would be independent of the number of processors. This test uses `mpi_sendrecv`.

Figure 4, page 7, presents the performance data for this test. For all 3 message sizes, the T3E shows the best performance and scalability. For an 8 byte message, the NT cluster scales and performs nearly as well as the T3E. Notice that for a 1 Mbyte message, the T3E, the SP and the Origin scale significantly better than the two clusters.

Test 4: The Naive Broadcast

For the naive broadcast test, each processor sends a message to processor 0. In a perfectly scalable computer the execution time for this test would increase linearly with the number of processors.

Figure 5, page 8, contains the results for this test. No data are provided for a 1 Mbyte message because of long execution times and lack of sufficient machine time. For this test, the NT cluster shows the best performance and scalability for an 8 byte message, whereas the T3E shows the best performance and scalability for a 10 Kbyte message.

Test 5: The Binary Tree Broadcast

The goal of this test is to evaluate the ability of the machines with communications via a binary tree broadcast. This communication pattern is often used to implement other common operations such as reduce, gather and scatter [8, 13]. In a perfectly scalable computer the execution time for this test would increase logarithmically with the number of processors. When $p = 2^n$, the MPI implementation of this test for broadcasting a message a can be described as follows. Using `mpi_send`, processor 0 sends a to processor $p/2$. Using `mpi_recv`, processor $p/2$ receives a from processor 0. Then, processors 0 and $p/2$ send a to processors $p/4$ and $3p/4$, respectively. This pattern is repeated until all processors have received a.

Figure 6, page 9, presents the results of this test. For all 3 message sizes, the T3E performs and scales the best. Notice that the NT cluster performs and scales as well as the T3E for an 8 byte message. For a 1 Mbyte message, the T3E, the Origin and the SP scale significantly better than the 2 clusters.

Test 6: The All-To-All

The all-to-all test was selected to evaluate the scalability of the machines for communication with high contention. For this test, each processor receives a message from each of the other processors. In a perfectly scalable computer the execution time for this test would increase linearly to the number of processors.

Figure 7, page 10, presents the data of this test. No data are provided for a 1 Mbyte message because of memory limitations. For an 8 byte message, the T3E and the NT cluster perform and scale the best. For a 10 Kbyte message, the T3E, the SP and the Origin scale and perform significantly better than the two clusters.

4 Conclusion

This paper presents scalability and communication performance results for a cluster of PCs running Linux with the GM communication library, a cluster of PCs running Windows NT with the HPVM communication library, a Cray T3E-600, an IBM SP and a Cray Origin 2000. Six communication tests using MPI routines [15] were run 16, 32, ..., 96 processors for the SP and 16, 32, ..., 128 processors for the other machines. Exactly the same algorithm was used for each test on each machine except for the synchronization barrier test where vendor implementations were used. The tests were chosen to represent commonly used communication patterns with low contention to communication patterns with high contention.

For most of the tests the T3E provides the best performance and scalability. For an 8 byte message the NT cluster performs about the same as the T3E for most of the tests. For all the tests but one, the T3E, the Origin and the SP outperform the two clusters for the largest message size (10 Kbytes or 1 Mbyte).

5 Acknowledgments

We would like to thank SGI for allowing us to use their Origin 2000 and T3E-1200 located in Eagan, Minnesota.

We would like to thank the National Center for Supercomputing Applications at the University of Illinois in Urbana-Champaign, Illinois, for allowing us to use their NT Supercluster.

We would like to thank University of New Mexico for access to their Maui High Performance Computing Center and their Albuquerque High Performance Computing Center. This work utilized the UNM-Alliance Roadrunner Supercluster at the AHPCC and the UNM-AFRL IBM SP systems at the MHPCC.

References

[1] AHPCC Linux Supercluster - Web Server. http://www.alliance.unm.edu/.

[2] Cray Research Web Server. http://www.cray.com.

[3] IBM Web Server. http://www.austin.ibm.com.

[4] Maui High Performance Computing Center Web Server. http://www.mhpcc.edu.

[5] NCSA NT Cluster - Web Server. http://www.ncsa.uiuc.edu/General/CC/ntcluster/.

[6] Origin Servers. Technical report, Silicon Graphics, April 1997.

[7] A. Anderson, J. Brooks, C. Grassl, and S. Scott. Performance of the CRAY T3E Multiprocessor. In *Proceedings of SC97*, 1997.

[8] M. Barnett, S. Gupta, D. G. Payne, L. Shuler, R. van de Geijn, and J. Watts. Building a High-Performance Collective Communication Library. In *Supercomputing'94*, Washington D. C., November 1994. IEEE Computer Society Press.

[9] R. A. Bhoedjang, T. Rhl, and H. E. Bal. User-level network interface protocols. *IEEE Computer*, 31(11):53–60, November 1998.

[10] J. Fier. *Performance Tuning Optimization for Origin 2000 and Onyx 2*. Silicon Graphics, 1996. http://techpubs.sgi.com.

[11] G. R. Luecke, B. Raffin, and J. J. Coyle. Comparing the Scalability of the Cray T3E-600 and the Cray Origin 2000 Using SHMEM Routines. *The Journal of Performance Evaluation and Modelling for Computer Systems*, December 1998. http://hpc-journals.ecs.soton.ac.uk/PEMCS/.

[12] G. R. Luecke, B. Raffin, and J. J. Coyle. Comparing the Communication Performance and Scalability of a Linux and a NT Cluster of PCs, a Cray Origin 2000, an IBM SP and a Cray T3E-600. Extended version. August 1999. http://http://www.public.iastate.edu/~grl/publications.html.

[13] G. R. Luecke, B. Raffin, and J. J. Coyle. The Performance of the MPI Collective Communication Routines for Large Messages on the Cray T3E600, the Cray Origin 2000, and the IBM SP. *The Journal of Performance Evaluation and Modelling for Computer Systems*, July 1999. http://hpc-journals.ecs.soton.ac.uk/PEMCS/.

[14] S. L. Scott and G. M. Thorson. The Cray T3E Network: Adapatative Routing in a High Perfromance 3D Torus. In *HOT Interconnects*, Stanford University, August 1996.

[15] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI, The Complete Reference*. Scientific and Engineering Computation. The MIT Press, 1996.

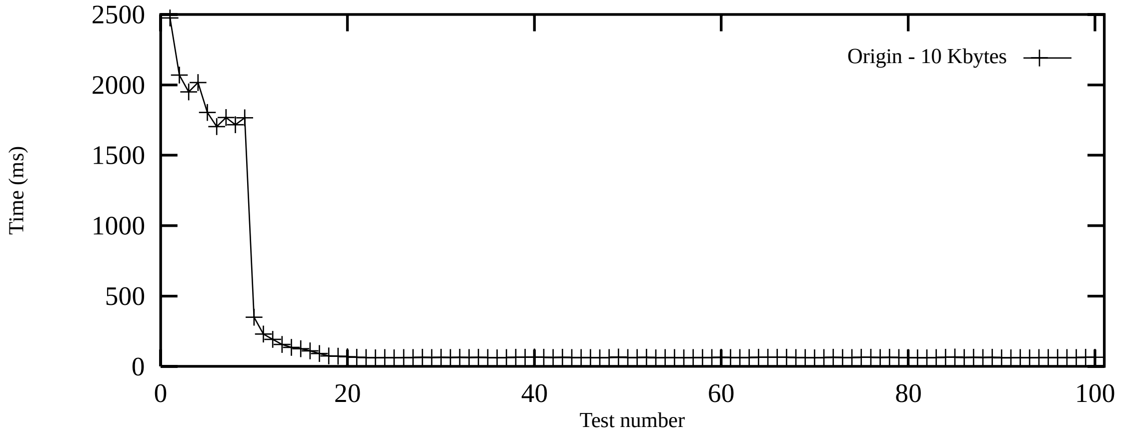


Figure 1. Execution times on the Origin 2000 for 100 executions of the all-to-all test with 112 processors and a 10 Kbyte message.

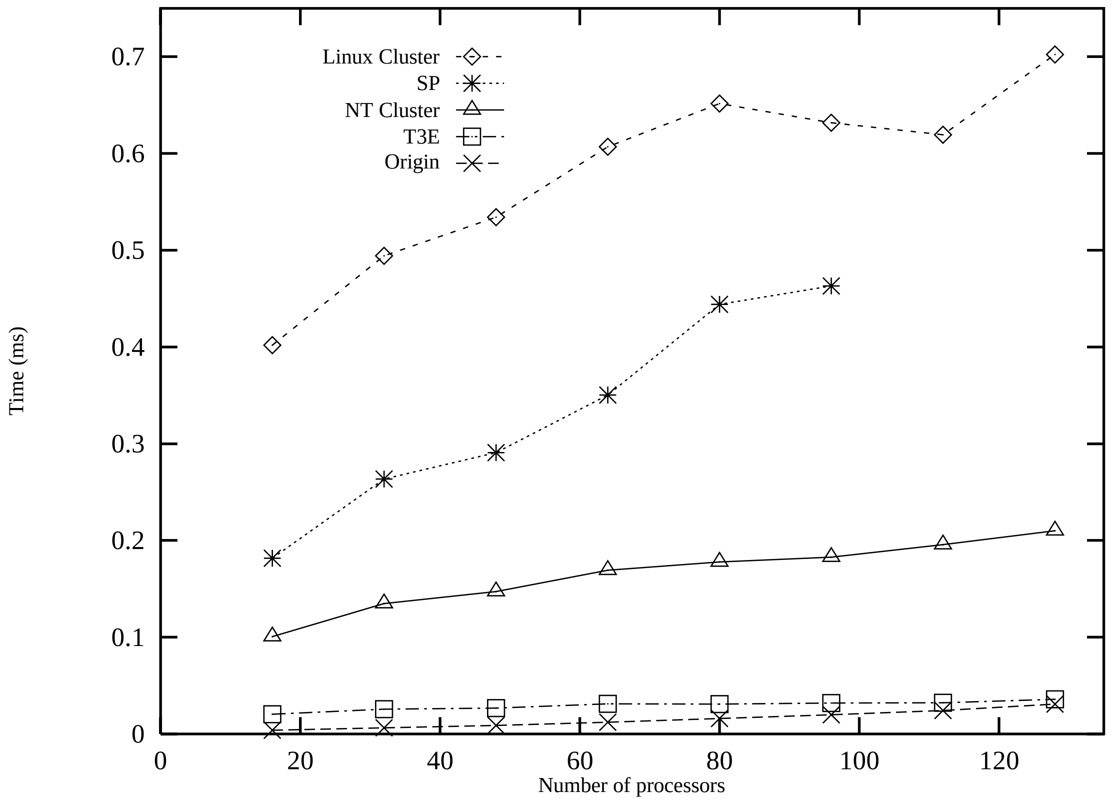


Figure 2. Test 1: mpi_barrier

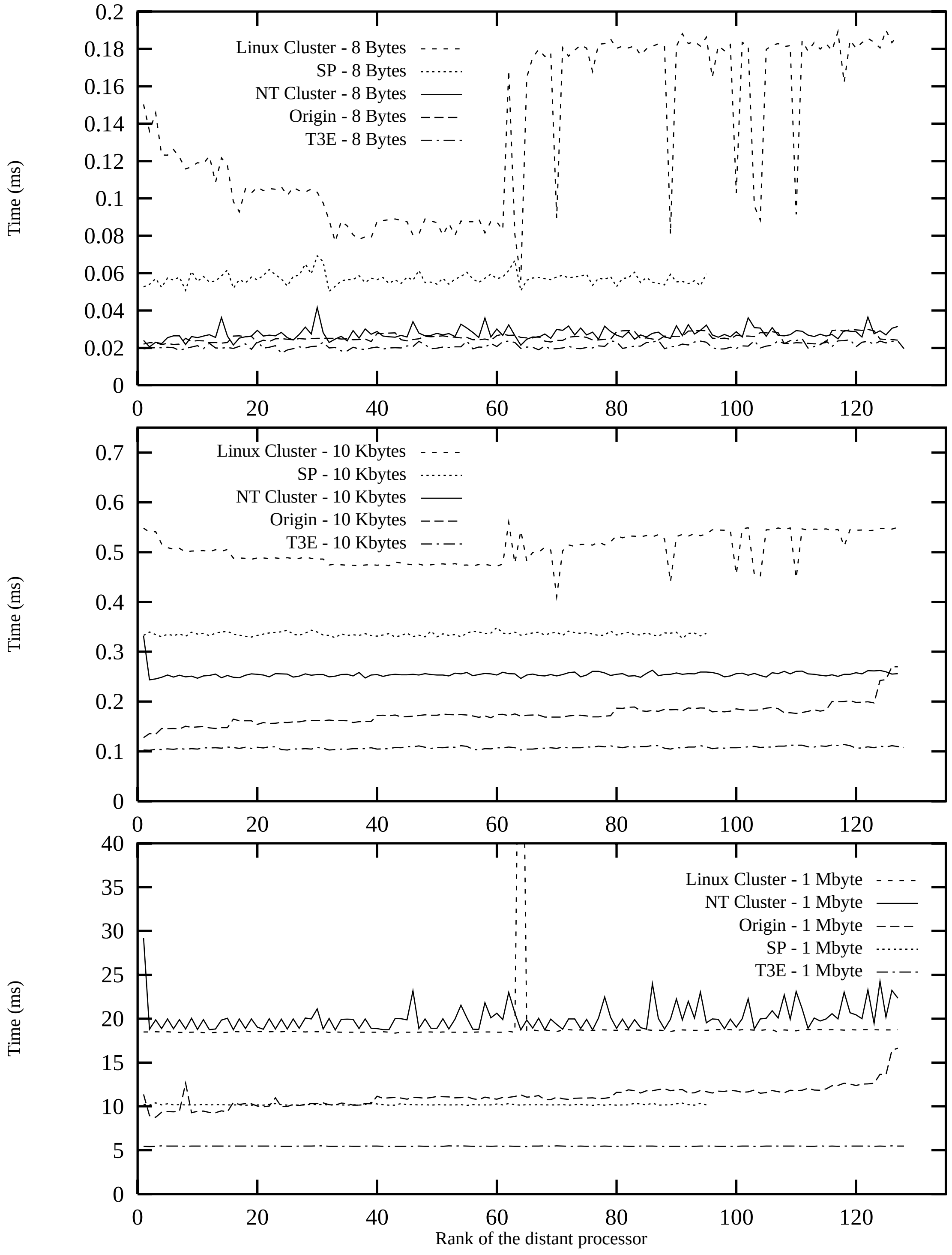


Figure 3. Test 2: Ping-Pong Between Processors. For clarity, only the lines connecting data points are shown.

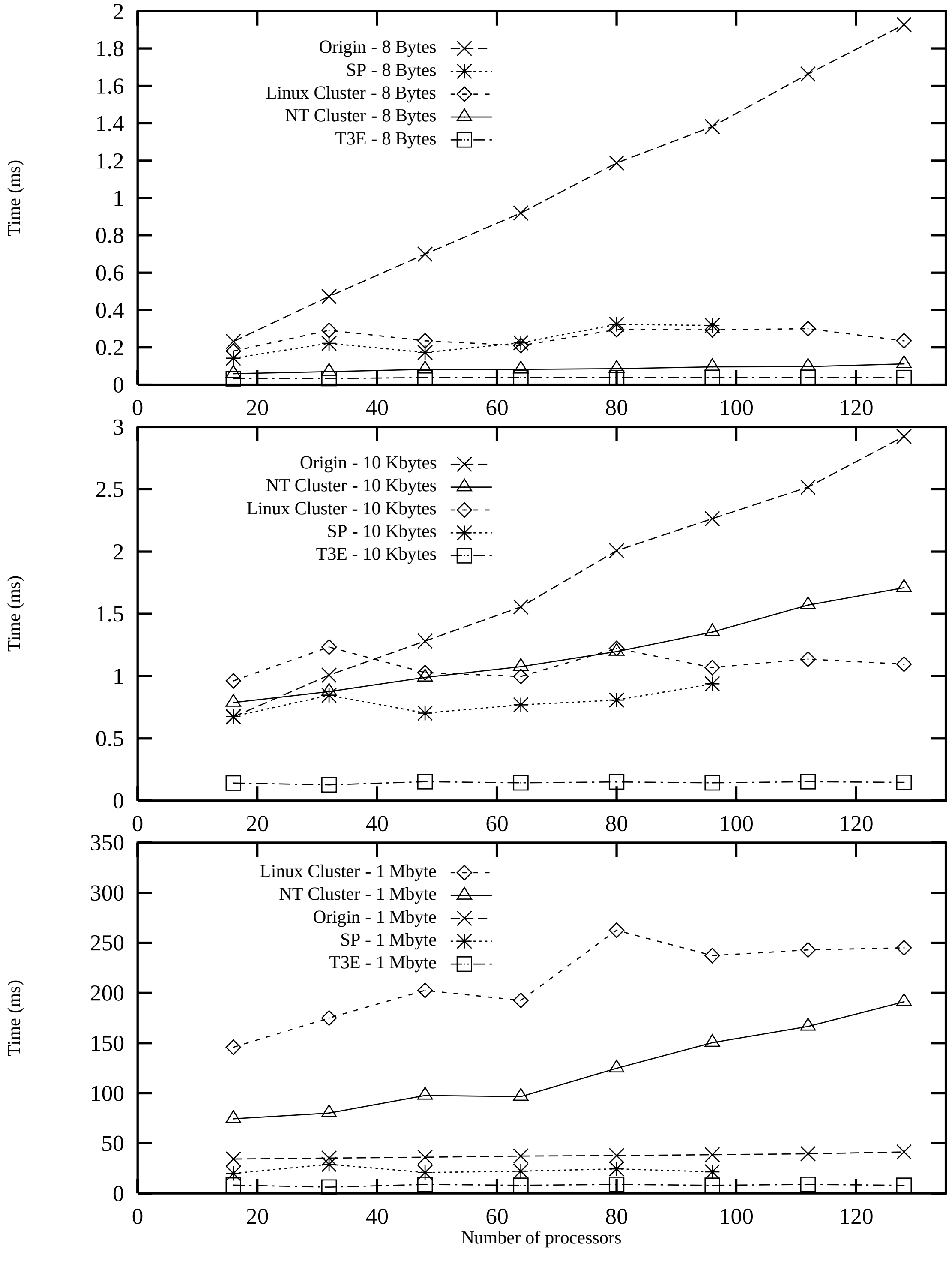


Figure 4. Test 3: Right Shift

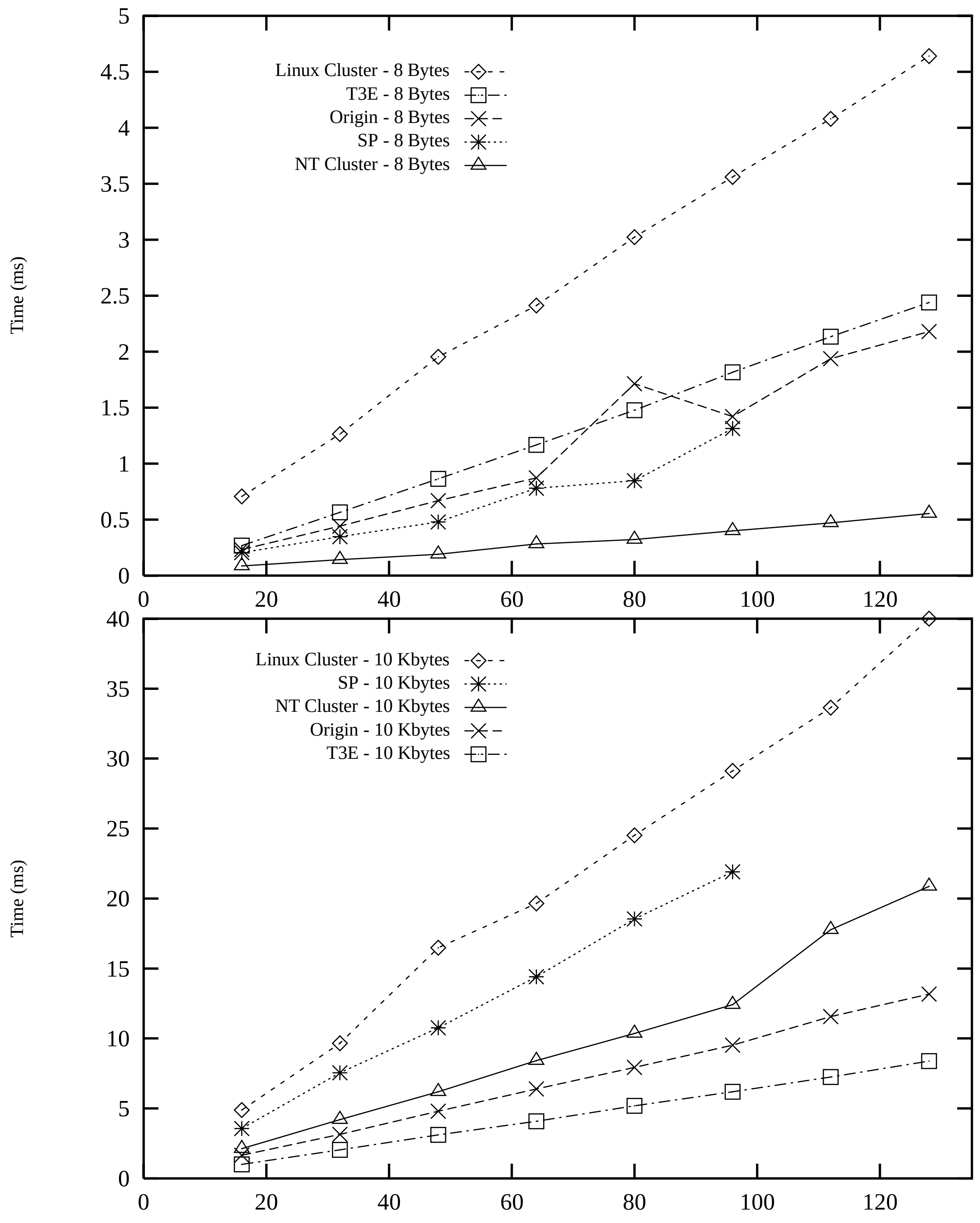


Figure 5. The Naive Broadcast in Test 4

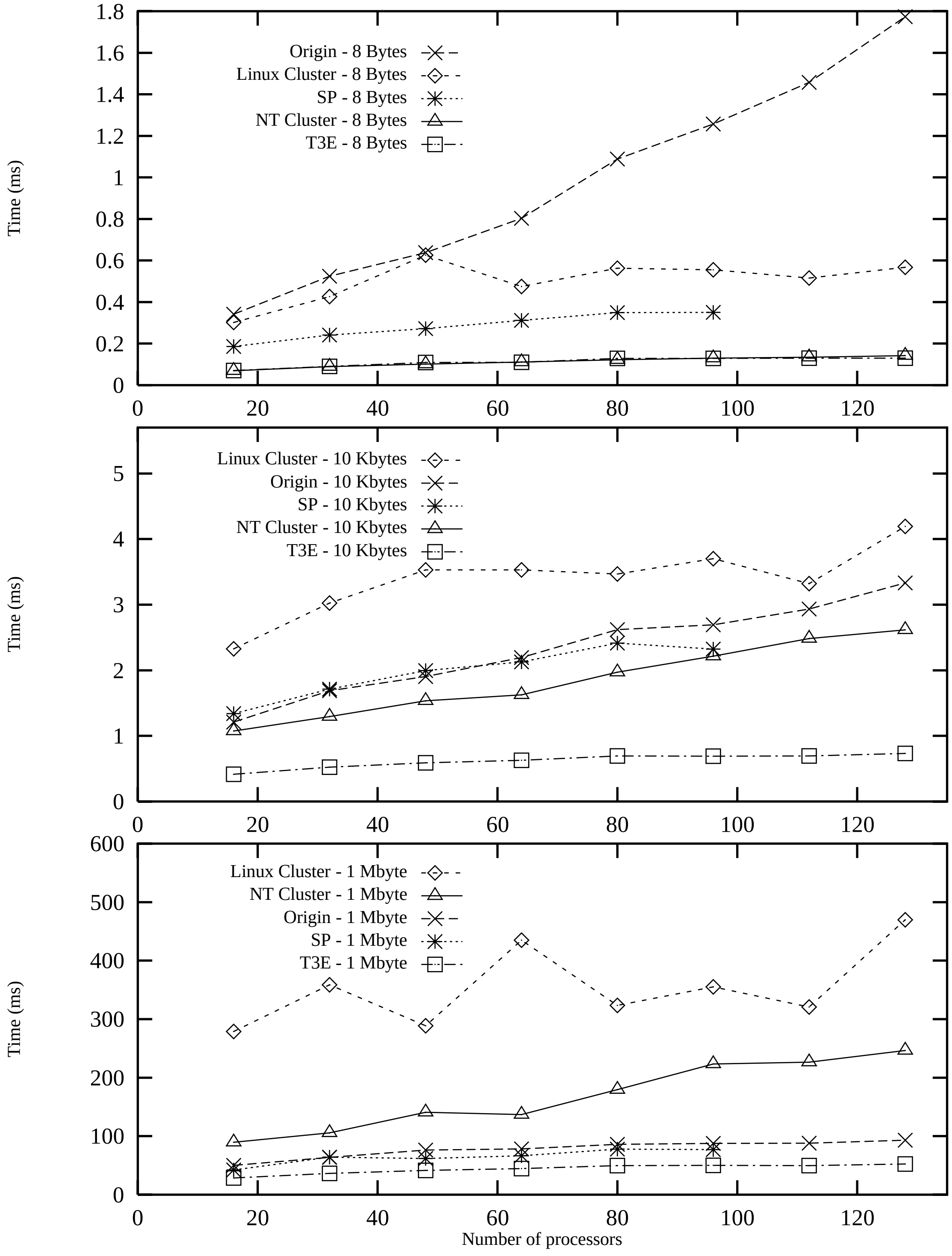
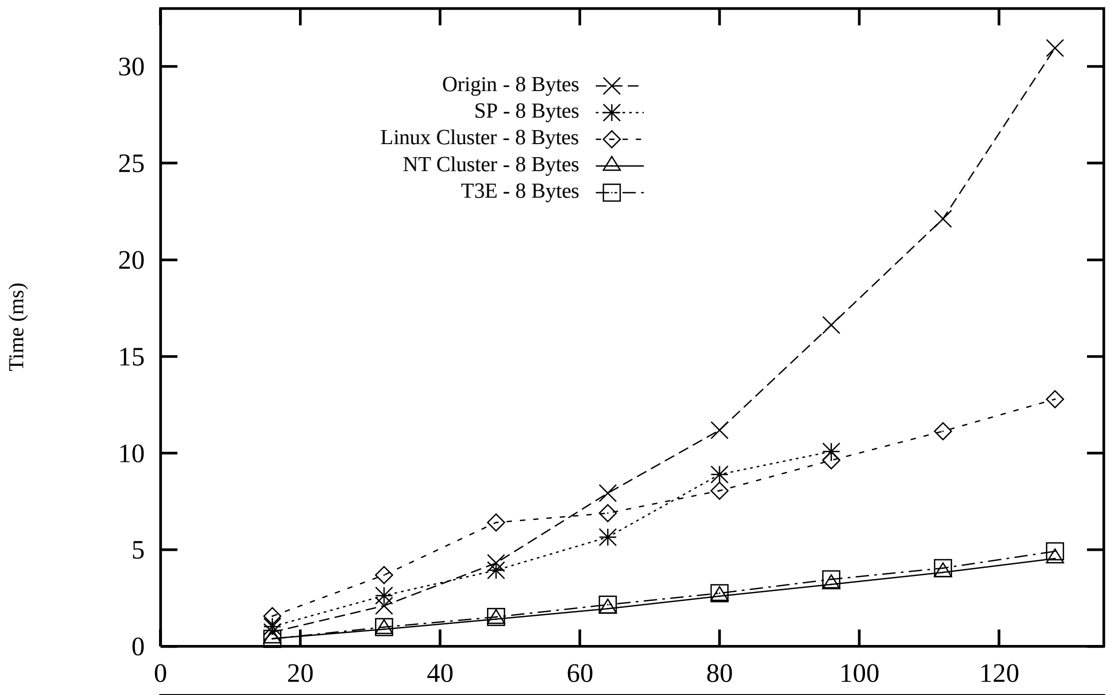


Figure 6. The Binary Tree Broadcast in Test 5



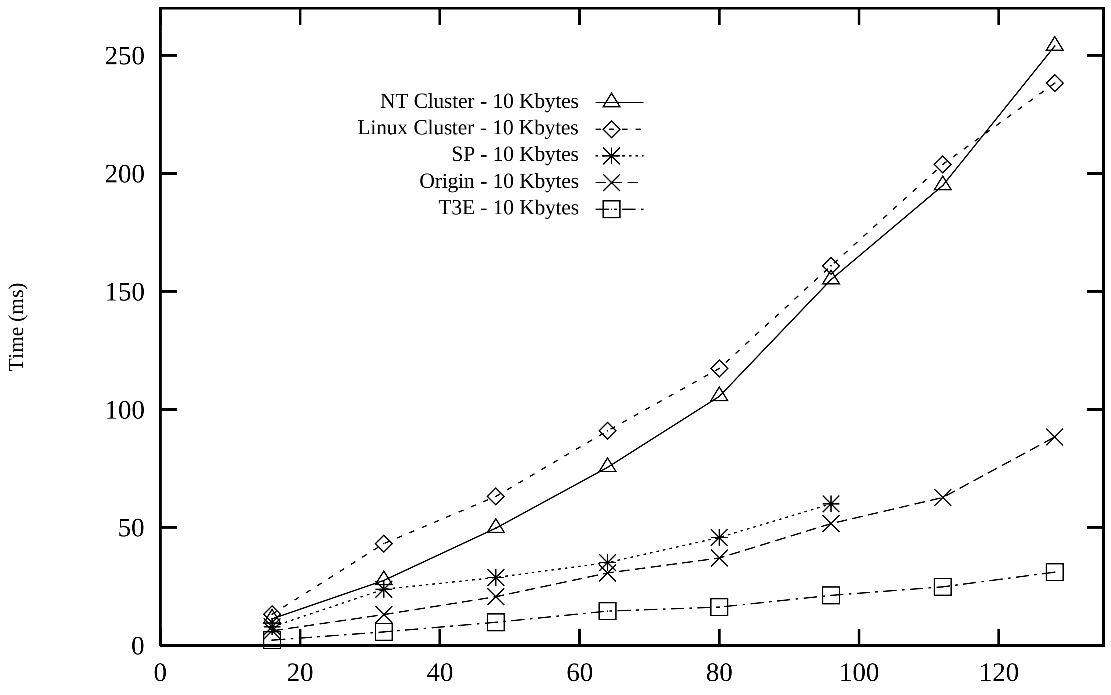


Figure 7. The All-To-All in Test 6