

MPICH-G2 Implementation of an Interactive Artificial Neural Network Training

D. Rodríguez¹, J. Gomes², J. Marco¹, R. Marco¹, and C. Martínez-Rivero¹

¹ Instituto de Física de Cantabria (CSIC-UC), Avda. de los Castros s/n, 39005 Santander, Spain

² Laboratório de Instrumentação e Física de Partículas, Lisbon, Portugal

Abstract. Distributed Training of an Artificial Neural Network (ANN) has been implemented using MPICH-G2, and deployed on the testbed of the european CrossGrid project. Load balancing, including adaptive techniques, has been used to cope with the heterogeneous setup of computing resources. First results show the feasibility of this approach, and the opportunity for a Quality of Service framework. To give an example, a reduction in the training time from 20 minutes using a single local node down to less than 3 minutes using 10 nodes distributed across Spain, Poland, and Portugal, has been obtained.

1 Introduction

The Grid [1] provides access to large shared computing resources distributed across many local facilities.

MPICH-G2 [2] is a Grid enabled implementation of MPICH [3]; it uses the Globus Toolkit 2 [4] to access Grid resources and perform tasks such as authentication, authorization, process creation and communications.

Training of Artificial Neural Networks (ANN) algorithms, depending on the size of the samples used, and the ANN architecture, may require large computation time in a single workstation, preventing an interactive use of this technique for data-mining.

As an example, the ANN used in the search for the Higgs boson using the data collected by the DELPHI detector [5] at the LEP accelerator at CERN, required from several hours to days to complete the training process using about one million simulated events for a simple two hidden layer architecture with about 250 internode weights. This prevents the final user from working in a real interactive mode while looking for an optimal analysis.

A previous work [6] has shown that a usual ANN training algorithm, BFGS [7], can be adapted using MPI to run in distributed mode in local clusters. A reduction in the training time from 5 hours to 5 minutes was observed when using 64 machines on a cluster [8] built using linux nodes connected by fast ethernet.

In this paper a first experience in a Grid framework is reported. The adaptation is based on the use of MPICH-G2, and has been deployed in the testbed of the european project CrossGrid [9].

The scheme of the paper is as follows: first the computing problem is described, including a reference example, and the adaptation using MPICH-G2 in a local cluster (section 2). First results in the Grid environment are given in section 3, where the load balancing mechanism is discussed. Section 4 presents the conclusions and a description of the future work.

2 Distributed training of an Artificial Neural Network using MPICH-G2 in a local cluster

Parallel ANN training is a topic that has been studied for a long time. A good review can be found in [10]. The program used in this work is based on the one already cited before [6].

The objective of the training is the minimization of the total error, given by the sum of errors for all events, each one defined as the quadratic difference between the ANN output computed corresponding to each event and the 1 or 0 value corresponding to a signal or background event. The minimization procedure is iterative, each iteration being called an epoch. The BFGS [7] gradient descent method has been used in the ANN training. In this method, errors and gradients are additive in each epoch. The results obtained in each slave can be added and transmitted to the master. This is a very good characteristic for a parallelization in data strategy.

The parallel training algorithm goes as follows:

1. The master node starts the work reading the input parameters for the ANN architecture, and setting the initial weights to random values.
2. The training data is split into several datasets of similar size (taking into account the computing power of each node to assure load balancing by data partitioning) and distributed to each slave node.
3. At each step, the master sends the weight values to the slaves, which compute the error and the gradient on the partial dataset and return them to the master; as both are additive, total errors are calculated by the master that prepares the new weights along the corresponding direction in the multidimensional space and update the weights. The error computed in a separated test dataset is used to monitor the convergence.
4. The previous step (an epoch) is repeated until a convergence criteria is fulfilled or a predetermined number of epochs is reached. The master returns as output the ANN weights, to prepare the corresponding multidimensional function.
5. The whole training can be repeated with different initial random weights to prevent bad results due to local minima. The final error can be used to select the best final ANN function if different minima are found.

A first step was to move from MPICH with the ch_p4 device to MPICH-G2, i.e. MPICH with the globus2 device. This implies the use of the Globus Toolkit. The Globus Toolkit is a collection of software components for enabling Grid computing: security, resource allocation, etc.

Number of Slaves	ch-p4 (s)	mpich-g2 (s)
1	11290	11242
8	1435	1448
16	720	736

Table 1. Performance comparison with the previous version of the program (using ch_p4).

Number of Slaves	Total time (s)
1	11242
2	5652
4	2827
8	1448
12	966
16	736
20	609
24	527
28	455
32	401

Table 2. Results in local cluster up to 32 slaves.

The comparison of the new program results, using the globus2 device, with those obtained with the previous version (the ch_p4 one) is done on a smeared simulated sample with a total of 412980 events, comparing the program execution time for a 1000 epoch training of an ANN with the 16-10-10-1 architecture. In table 1 one can see that performances are very similar.

The tests in a cluster environment were done at the IFCA's cluster at Santander [8]. It has 80 dual Pentium III IBM x220 servers (1.26 GHz, 512 Kb cache, 640 MB RAM with 36 GB SCSI and 60 GB IDE disks) running RedHat Linux 7.2. For these tests we used a maximum of 32 slaves.

To be able to change the number of participating slaves, all the data was distributed amongst the nodes. In this way each slave can access the data it needs independently of their number.

The tests using a single processor in each node show a time reduction from about three hours in a single node to near seven minutes with 32 nodes, considering the same 16-10-10-1 architecture trained over 1000 epochs (see table 2). The speedup in the local cluster is plotted in figure 1.

3 From Cluster to Grid Computing

The testbed used in the tests, described in detail in [11], includes 16 sites distributed across 9 different european countries, interconnected through the academic network Géant [12] and the corresponding national networks. Access to

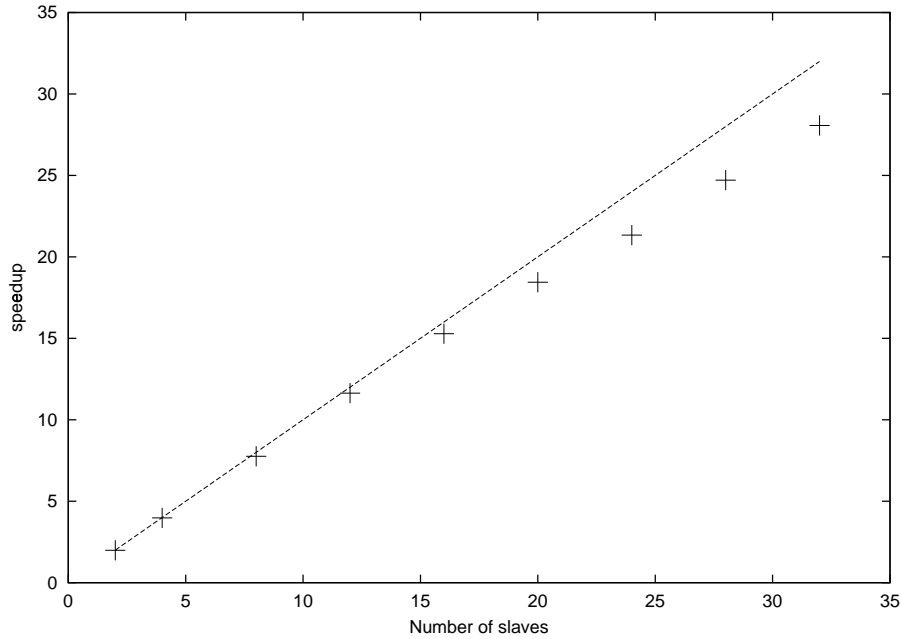


Fig. 1. Speedup in a local cluster for a 16-10-10-1 architecture using a single processor in each node

the testbed is granted through the use of certificates to users and its inclusion in a Virtual Organization. For this first study, only few nodes distributed at several sites in Spain, Portugal and Poland have been used, in order to reduce the complexity of understanding the results obtained, while preserving the distributed nature of the experiment.

Topology aware collective operations as discussed in [13] can provide significant performance improvements over a non topology aware approach in the case of MPI applications in the Grid. In our case the MPI.Reduce operation used for the addition of errors and gradients in each epoch could be the critical one.

3.1 Security

Security issues are extremely challenging when moving to a Grid environment. In this transition one has to move from a locally administered environment to a heterogeneous WAN with multiple administrators. Mutual trust between the users and the sites is essential. The authentication of users and systems is done through a public key infrastructure (PKI) based on X.509 certificates that must be recognized and accepted by all intervening parties. The authorization is performed through a *Virtual Organizations* (VO) mechanism thus allowing users to access resources across sites with the same authentication credentials.

The CrossGrid virtual organization was used to perform the Grid tests. A VO as defined in [14], is a set of individuals and/or institutions defined by several rules for sharing computing resources: machines, software, data, etc. In our tests all the security uses this grid certificates mechanism. The security infrastructure thus relays in the testbed (see [11]).

3.2 Data stripping and transfer

An important issue is the availability of distributed replica for the training data, as its proximity in access time to the computing resources is a key point to reduce the waiting time. The Grid framework provides several interesting possibilities on data replication and distributed data access (see for example [15], [16] and [17]). For this first deployment the data has been directly transferred to the different nodes and the corresponding computing elements. In this note we have placed our data inside the computing elements, in order to minimize its exchange.

3.3 Load Balancing

Having into account that the speed of a training step is the speed of the slowest slave the need for a good load balance is essential.

The used dataset contains a total of 412980 events and has the following components: 4815 signal (Higgs bosons) events and two different kinds of background events: 326037 WW events, and 82128 QCD events. For our tests we further divided each of the subsets in 80 slices, and replicated them across the participating nodes.

In a configuration using a 1.26 GHz PIII master, and 5 slaves (4 dual 2.4 GHz P4 Xeon in a cluster in Poland and one 1.7 GHz P4 in Portugal) , we observed that an equal distribution of events between them resulted in a heavy performance penalization due to the slowest slave. This one spends –and thus increases the total time– near a 40% more time than the fastest one per training epoch.

Introducing an automatic data distribution amongst the slaves based on a weighting factor that, in a first approach, was chosen to be the CPU speed, the 1.7 GHz node continues to be the slowest slave, but the delay per epoch is reduced to only a 5% time increase with respect to the fastest slave. One can compare the results obtained with and without load balance in table 3. These results show how a balanced training reduces the time spent per epoch in the slowest node, increasing thus the overall speed of the program.

A further improvement would be to benchmark the nodes in order to refine the weighting factor. Preliminary results running a version of our program with only one slave and the full data sample show sizeable improvements.

It is worth noticing that both methods imply an a-priori knowledge of the nodes where the application will be run that will not be available in a Grid production environment. This knowledge should be substituted by the information provided by Grid utilities.

Node	Not balanced		Balanced	
	Num. events	Time per epoch (s)	Num. events	Time per epoch (s)
Node 1	82651	0.757	87677	0.808
Node 2	82576	0.757	87736	0.816
Node 3	82576	0.758	87760	0.825
Node 4	82576	0.759	87764	0.813
Node 5	82061	1.07	62061	0.848

Table 3. Load balanced gain in the CrossGrid testbed for an ANN train epoch. The times are the mean of the times consumed in each epoch in the corresponding slave, not the total application time divided by the number of epochs.

Master	slaves@krakow	slaves@lisbon	Total time (s)
IFCA	9	1	218.4
IFCA	7	1	230.8
IFCA	5	1	258.9
IFCA	6	0	254.0
IFCA	4	0	312.0
IFCA	2	0	525.8
INP	2	0	452.3

Table 4. Performance comparison with different testbed nodes.

One can compare the results obtained using different testbed configurations in table 4. In all cases a node placed at IFCA in Spain acted as a master except in the last case where everything was done inside the same cluster (INP at Krakow in Poland); one can see then how the computation time decreases with respect to the previous line in the table due to the absence of network delay. To compare with, the same program needs 1197 seconds to run in a dual PIII 1.26 GHz. So, even in the Grid environment, you can get a clear time improvement when parallelizing the ANN training. The training time is reduced from 20 minutes in the local node, to a bit more than 3 minutes with 10 slaves.

A first attempt to use an adaptative technique to cope with changing testbed conditions has been performed. As the program is meant to be part of an interactive training process we consider that the loss of some events is not critical. Thus, when a node is slowing significantly the training process we can reduce the number of events it uses in the training. Anyhow, the total amount of events lost should not exceed a certain percentage of the total events, and the error display (figure 2) is a good monitoring tool. This can result in a useful feature as changing conditions in the network traffic or in the machines load would severely damage performance even having a good a-priori load balancing. This feature can be disabled if desired by the user, and was only used for the last test referred in this note.

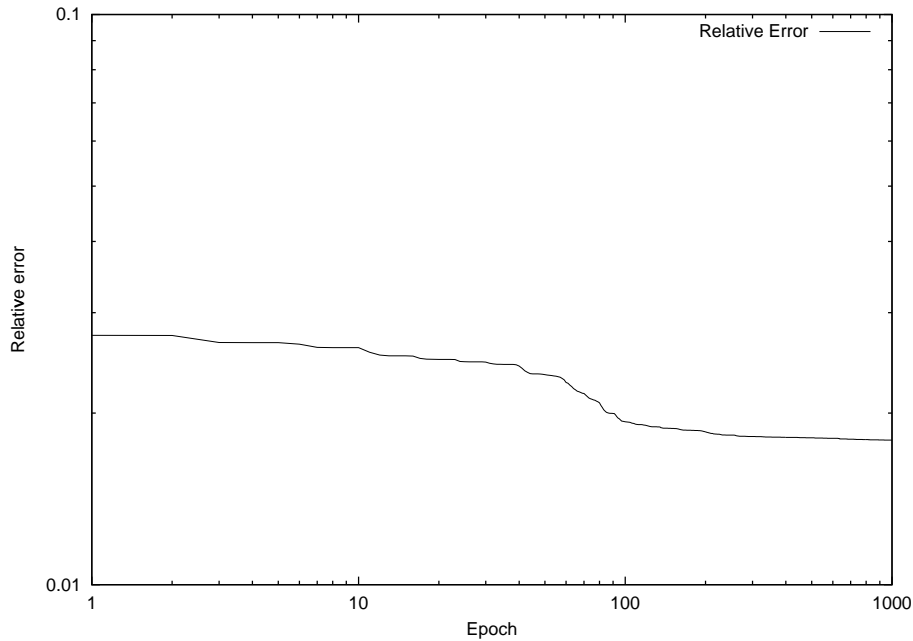


Fig. 2. Error display of the ANN program. Both axis are in logarithmic scale.

4 Conclusions and Future Work

The results of this paper show the feasibility of running a distributed neural network in a local cluster reducing the wait time for a physicist using this tool from hours to minutes. They also indicate that this approach can be extended to a Grid environment, with nodes distributed across a WAN, if the latency is low enough. The time reduction obtained shows that the resources made available by the Grid can be used to perform an interactive ANN training. The nature of the ANN training problem let us implement an especial dynamic load balancing approach, based on the reduction of the number of events, that might not be applied for much other problems. A more general solution should be researched in the future.

Along these tests we noticed a strong need for a quality of service approach in Grid applications, specially for interactive ones. The reservation of resources (network bandwidth, processing and memory capacity) is critical for a satisfactory execution of this sort of applications (of which our parallel ANN training is a good example). Although we have the possibility to query the resources availability when starting execution, not having a guarantee on the availability of resources during execution can severely damage the performance.

Parallel (MPI) applications can be a very challenging case for Quality of Service frameworks. In our case, the master cannot compute the weights for the new

epoch until it receives the errors and gradients from all the slaves. So computation is stopped (and time is lost) while waiting for communications to complete. On the other hand, while the master or the slaves are computing, there are no communications; so we have a very irregular communication pattern. There are peaks of communications at the beginning and at the end of each training epoch, and no communications at all during the new epoch error computation.

Furthermore, communications can involve many processes making the problem even more complex. Not only the fact that more computing nodes increase the number of sites sending packets, but also the waiting time between epochs is reduced as the number of computing nodes increase, reducing the time between communication peaks, and potentially increasing the contention problem in shared networks.

A prototype implementation of a Quality of Service architecture for MPI programs can be found in [18]. We think this is a very interesting topic that can be critical for the success of interactive applications on the Grid. We are working for a Quality of Service for the network in the CrossGrid testbed.

Some further improvements that are being considering are:

- Improving the event reduction mechanism.
- Usage of Grid information utilities.
- Integration in a portal.

5 Acknowledgements

This work has been mainly supported by the European project CrossGrid (IST-2001-32243). We would like to thank in particular to all the testbed sites for offering the possibility to run our tests.

References

1. I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.
2. N. Karonis, B. Toonen, and I. Foster, *MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface*, Journal of Parallel and Distributed Computing (JPDC), to appear, 2003.
3. MPICH. <http://www-unix.mcs.anl.gov/mpi/mpich>
4. The Globus Project. <http://www.globus.org>
5. DELPHI collaboration. <http://delphiwww.cern.ch/>
6. O. Ponce et al. *Training of Neural Networks: Interactive Possibilities in a Distributed Framework*. In D. Kranzlmüller et al. (Eds.) *9th European PVM/MPI*, Springer-Verlag, LNCS Vol. 2474, pp. 33-40, Linz, Austria, September 29-October 2, 2002.
7. Broyden, Fletcher, Goldfarb, Shanno (BFGS) method. For example in *Practical Methods of Optimization* R.Fletcher. Wiley (1987)
8. Santander Grid Wall. <http://grid.ifca.unican.es/sgw>
9. CrossGrid European Project (IST-2001-32243). <http://www.eu-crossgrid.org>
10. Manavendra Misra. *Parallel Environments for Implementing Neural Networks*. Neural Computing Survey, vol. 1., 48-60, 1997.

11. J. Gomez et al. *First Prototype of the CrossGrid Testbed*. Presented at the 1st Across Grids Conference. Santiago de Compostela Feb. 2003.
12. Géant. <http://www.dante.net/geant/>
13. N. Karonis et al. *Exploiting hierarchy in parallel computer networks to optimize collective operation performance*. In Proceedings of the 14th International Parallel and Distributed Processing Symposium, 2000.
14. I. Foster, C. Kesselman, S. Tuecke. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. International J. Supercomputer Applications, 15(3), 2001.
15. H. Stockinger et al. File and Object Replication in Data Grids. Journal of Cluster Computing, 5(3)305-314, 2002.
16. A. Chervenak et al. *Giggle: A Framework for Constructing Scalable Replica Location Services*. In Proceedings of SC2002, Nov. 2002.
17. OGSA-DAI. Open Grid Services Architecture Data Access and Integration <http://www.ogsadai.org.uk/>
18. A. Roy et al. *MPICH-GQ: Quality-of-Service for Message Passing Programs*. Proceedings of SC2000. Dallas, Nov. 2000.