Neural Comput & Applic (2012) 21:535–542
DOI 10.1007/s00521-011-0544-3

ORIGINAL ARTICLE

Behavior analysis of neural network ensemble algorithm on a virtual machine cluster

Cesar Fernández · Carlos Valle · Francisco Saravia · Héctor Allende

Received: 24 March 2010 / Accepted: 31 January 2011 / Published online: 18 February 2011

Abstract Ensemble learning has gained considerable attention in different learning tasks including regression, classification, and clustering problems. One of the drawbacks of the ensemble is the high computational cost of training stages. Resampling local negative correlation (RLNC) is a technique that combines two well-known methods to generate ensemble diversity—resampling and error negative correlation—and a fine-grain parallel approach that allows us to achieve a satisfactory balance between accuracy and efficiency. In this paper, we introduce a structure of the virtual machine aimed to test diverse selection strategies of parameters in neural ensemble designs, such as RLNC. We assess the parallel performance of this approach on a virtual machine cluster based on the full virtualization paradigm, using speedup and efficiency as performance metrics, for different numbers of processors and training data sizes.

Keywords Ensemble learning · Artificial neural networks · Virtualization · Multicore processor · Parallel algorithms

C. Fernández · C. Valle (✉) · F. Saravia · H. Allende
Department of Computer Science, Universidad Técnica Federico Santa María, CP 110-V Valparaíso, Chile
e-mail: cvalle@inf.utfsm.cl

C. Fernández
e-mail: cesferna@inf.utfsm.cl

F. Saravia
e-mail: fsaravia@inf.utfsm.cl

H. Allende
e-mail: hallende@inf.utfsm.cl

H. Allende
Faculty of Engineering and Science, Universidad Adolfo Ibañez, Santiago, Chile

1 Introduction

In the last years, model combining or ensemble modeling has become a topic of interest in the artificial neural networks field [1, 15, 16, 26]. The basic idea of this multi-model approach is to select a set of N predictors $S = \{f_0, f_1, \ldots, f_{N-1}\}$ and build an aggregation function F using an operator that combines the individual decisions.

For classification problems, the most common aggregation function F is the majority voting, while, for regression, the most common one is the weighted average of the individual outputs, where (given $\sum w_i = 1$)

$$F(x) = \sum_{i=0}^{N-1} w_i f_i(x). \tag{1}$$

Both theoretical and empirical findings have suggested that the combination of different models can be an effective way to obtain better predictive performance compared with the use of individual predictors, especially when the combination of models exhibits a sort of heterogeneity or *diversity*. For regression settings, the error e_i of each machine that is usually calculated as the squared difference between the output of the machine f_i and the actual output y,

$$e_i = (y - f_i)^2, \tag{2}$$

where e_i is offset by the error of other machine, thus the ensemble output F will be more accurate than an individual machine output. To generate diversity on ensembles, several ways exist, for example, the use of different initialization of the weights and biases of individual neural networks, different training algorithms to adjust weights and biases, different architectures in each neural network, different training sets, and enhancement of the negative correlation of error among the learners.

Resampling local negative correlation learning (RLNC) [18] is an algorithm for regression settings that combines the two learning strategies indicated above, diversity and stability, obtained by resampling. This algorithm creates a kind of local negative correlation between the individual errors to get a set of diverse predictors. Resampling allows this base algorithm to control the impact of highly influential data points which in turns can improve its generalization error. The resulting approach can also be explained as a generalization of Bagging [3], in which each learner is no longer independent but locally coupled with other learners. RLNC has demonstrated to have competitive generalization ability when compared with well-known ensemble algorithms.

A parallel implementation of RLNC [22] reduces considerably the training process time, taking advantage of the underlying parallelism of the algorithm and the new generation of multicore processor architecture. In addition, it creates new possibilities to explode the potentials of calculus and parallel processing, but one of its limits is undoubtedly to obtain real benefits regarding speedup and performance metrics of this architecture. A high percentage of the most important high-performance computing (HPC) centers in the world have used this new architecture, and many others have begun to migrate from single-core processors to multicore processors [10].

Virtualization [8] consists in adding a software layer between the operating system and the hardware, denominated "Virtual Machine Monitor" (VMM). This intermediate layer allows the installation of diverse operating systems and allows the management operations of physical and logical resources to be transparent to each system. Virtualization has proliferated quickly in academic [13] and industry fields [19] because it has contributed to avoid the investments in new hardware, to improve the use of physical platform and to reduce the testing time and the production of new applications or legacy systems. The use of virtualization is diverse and extensive. It can be applied to a group of several operating systems in the same physical machine as well as to a whole-data center, including several hardware and software architectures, applications and networks. In a number of specialized researches, the level of communication and some measures of performance have been analyzed [6, 21, 24], but the impact on the performance is not clear when virtualization is introduced on parallel applications, making necessary to perform optimizations even in HPC environments [14].

This paper introduces an analysis of the behavior of RLNC using parallel performance metrics on a virtual machine cluster. To perform this analysis, a comparative study between virtual and physical machine cluster must be carried out, in which the parallel approach of RNLC is executed with the same configurations in both environments to determinate the effects on parallel performance introduced by virtualization.

This paper is arranged as follows: the next section describes the concepts of diversity and negative correlation to consecutively present the RLNC algorithm that combines both properties. Section 3 presents the parallel implementation for RLNC on a virtual environment. Section 4 presents the configuration of the testing of environment and the discussion of the experimental results. Finally, Sect. 5 presents the findings and future work.

2 Resampling local correlation learning (RLNC)

For estimating regression, a theoretical approach to measure diversity is the so-called *Ambiguity Decomposition* [5] of the quadratic loss of an ensemble F that is obtained using Eq. (1). This decomposition states that

$$\bar{e} = (y - F)^2 = \sum_{i=0}^{N-1} w_i (y - f_i)^2 - \sum_{i=0}^{N-1} w_i (F - f_i)^2. \tag{3}$$

The ensemble error of $\bar{e}$ can be divided into two terms: the first term reflects the weighted squared error of each machine, i.e., the weighted sum of individual errors on the target y. The second term is called *ambiguous term*, which corresponds to the error of each machine regarding the joint output F in the ensemble. It can be alternatively stated as [17]

$$(y - F)^2 = \sum_{i=0}^{N-1} w_i^2 (y - f_i)^2 + \sum_{i=0}^{N-1} \sum_{j \neq i} w_i w_j (f_i - y)(f_j - y). \tag{4}$$

As in the ambiguity decomposition, the first term measures the individual performance of the estimators, while the second one measures the error correlation between the different predictors. Considering an ensemble where all learners are uniformly weighted, from this decomposition it seems proper to train each learner $i = 0, \ldots, N-1$ with the training function

$$\tilde{e}_i = (y - f_i)^2 + \eta \sum_{j \neq i} (f_i - y)(f_j - y), \tag{5}$$

where $\eta > 0$ controls the balance between individual performance and the amount of diversity among individual learners. Ensemble diversity can be generated using a set of locally coupled learners. Each learner f_i is related to a reduced and fixed subset of other learners V_i through the definition of a linear neighborhood function of order v.

$$\psi(i,j) = 1 \Leftrightarrow (i - j) \bmod N \leq v \text{ or } (j - i) \bmod N \leq v, \tag{6}$$

In that way, the learner f_j belongs to the neighborhood V_i of f_i only if $\psi(i, j) = 1$. Geometrically speaking, the learners are disposed on a ring, in which two learners are neighbors if they are contiguous up to v steps.

The objective function (5) can be made local by restricting it to the neighborhood of the i-th learner

$$e_i^{\text{local}} = (y - f_i)^2 + \eta \sum_{j \in V_i} (f_i - y)(f_j - y). \tag{7}$$

The interesting result is that in [17] the authors affirm that training synchronously the set of learners with these local objective functions is independent of the neighborhood order v. This means that a minimal degree of overlapping ($v = 1$) between the learners is enough to propagate the information about the performance of each learner to the whole group.

The manipulation of the training data to be learned for each ensemble member is a commonly used method. Different learners are provided with different training examples or different training features to learn different "aspects" about the same task. In this family of ensemble algorithms, resampling methods have proved to be highly effective. In Bagging, each learner is provided with a set of patterns obtained by randomly resampling the original set of examples and then trained independently from the other learners. Despite the simplicity of this approach, the results of the algorithm are, in many cases, superior to more elaborated algorithms.

The important fact is that certain sampling schemes allow some points to affect only a subset of learners in the ensemble. Empirical evidence suggests that Bagging equalizes the influence of training points in the estimation procedure in such a way that highly influential points (the so-called leverage points) are down-weighted [11]. Since in most situations, leverage points are badly influential, Bagging can improve generalization by making robust an unstable base learner. From this point of view, resampling has an effect similar to *robust M-estimators*, where the influence of sample points is (globally) bounded using appropriate loss functions, for example Huber's robust loss or Tukey's bisquare loss.

Since in uniform resampling, all the points in the sample have the same probability of being selected, it seems counterintuitive that Bagging has the ability to selectively reduce the influence of leverage points. The explanation is in the nature of leverage points itself. Leverage points are usually isolated in the feature space while non-leverage points act in groups. To remove the influence of a leverage point, the elimination of this point from the sample is enough, but to remove the influence of a non-leverage point, the general removal of a group of observations is essential. Furthermore, the probability that a group of size k be completely ignored by Bagging is $(1 - k/m)^m$, which decays exponentially with k. For $k = 2$, for example $(1 - k/m)^m \sim 0.14$ while $(1 - 1/m)^m \sim 0.37$.

In RLNC [18], each learner works with a different set of training patterns obtained by randomly resampling the original set of examples. Resampling allows to restrict the influence of leverage points to only a subset of learners in the ensemble, while the training criterion (7) still encourages cooperation between learners. As depicted in Algorithm 1, at each iteration, each learner takes into account the performance of the group only regarding to its own set of training patterns. Then, the influence of each training pattern is restricted to the learners whose training set contains the point.

As stated before, η controls the predominance of the group performance with regard to the individual performance. It should be noted that if $\eta = 0$, Bagging is obtained, that is, the learners are trained independently without any information about the group performance. If $\eta > 0$, this information is incorporated to the estimation process and an explicitly cooperative ensemble is obtained.

In each iteration, the neural networks receive the information of its neighborhood to update the weights and biases in order to the minimize the objective function (8).

Algorithm 1 RLNC

```
1: Let $D = \{(x_i, y_i); i = 1, \ldots, m\}$ be a set of training patterns
2: Let $f_i$, $i = 0, \ldots, N - 1$ be a set of $n$ learners and $f_i^t$ the function implemented by the learner $f_i$ at time $t = 0, \ldots, T$
3: Let $V_i$ be the neighborhood of $f_i$
4: Obtain $f_i^0$ applying one epoch of a gradient descend algorithm to minimize the square error
5: Generate $n$ new samples $D^i$, $i = 1, \ldots, n$ sampling randomly with the replacement of the original set of examples $D$
6: for $t = 1$ to $Q$ do
7:     Perform one epoch on the learner $f_i$ with the learning function
           $e_i^t = (y - f_i)^2 + \eta \sum_{j \in V_i} (f_i - y)\left(f_j^{t-1} - y\right)$    (8)
       and the set of examples $D^i$
8: end for
9: Set the ensemble at time $t$ to be $F(x) = 1/n \sum_{i=0}^{n-1} f_i(x)$
```

It suggests that RLNC has a fine-grain parallelism approach due to the training process requires relatively small amounts of computational work among communication events.

3 RLNC on virtual multicore cluster

A parallel computer or cluster of computers, consists of a set of computers, generally homogeneous, interconnected by a communication channel and works as a single large computer [7] to implement programs that require large computing capabilities or require more computing units working concurrently. The cluster processing takes place in the compute nodes. Each node is an independent computation unit. The computation is performed using a master–slave scheme, in which the execution is performed by the slave nodes and the process of coordination among slaves is performed by the master. A master–slave processing scheme is used in the parallel implementation of RLNC, this master–slave processing is commonly used in Message Passing Interface (MPI) [12].

There are several ways to implement virtualization, one of the most known is *paravirtualization*, a concept introduced in [25]. Paravirtualization or ring scheme [2] consists of dynamically assigning execution privileges or rings to each virtual operating system, in which value zero has the most privileges, so that this value is used by the hypervisor, who supervises the execution of tasks and the management of resources (logical and physical) of all virtual machines. Another way is the *full virtualization using binary translation* which consists of implementing virtualization using a combination between binary translation and direct execution techniques. This approach translates the kernel code to replace non-virtualizable instructions with other instruction set that has the same effect in the virtual hardware, while the code level is directly executed on the processor. The combination of both techniques provides *full virtualization*, thus the guest OS is completely decoupled from the underlying hardware by the virtualization layer.

In this work, a virtual cluster environment is set up using a full virtualization approach with VirtualBox [23] as virtual machine monitor, to create a flexible cluster setting through a set of virtual machines. This set supports a neural network ensemble, in which each learner, composed by three layers with five sigmoidal hidden units, is trained with standard backpropagation.

The algorithm includes the performance of several activities, such as the partitioning of data for training and testing, the resampling, the initialization of all neural networks in the ensemble, and the training of each neural network using the Eq. (7) as the training rule. The *diversity* is enhanced with the receiving information from the outputs of each learner that belongs to its neighborhood (V_i), where the training process of each learner is carried out with a bootstrap sample from the initial data set. The algorithm uses a *ring structure* for representing the interactions of each neural network with its neighborhood by sending and receiving messages. This operation is performed in parallel by all neural networks. The communications between each neural network and its neighbors takes place at this point.

At a bigger size of v, the communication time increases until to reach the worst performance case, when $v = \lfloor \frac{N}{2} \rfloor$, at this point (all to all), broadcast is performed. This produces an effect of "cloud of messages", thus a neighborhood with order $v = 1$ is selected, because it allows the minimization of the communication time. Given the previously described features, the algorithm is presented as follows:

Algorithm 2 RLNC Training of *i*-th learner

```
1: Define $V_i$ as described in equation (6)
2: Load $D^i$ and $D^J$ $\forall j \in V_i$
3: for $t = 1$ to $T$ do
4:   for $j \in V_i$ do
5:     SEND $f_i(x)$ $\forall x \in D^j$ to learner $j$
6:   end for
7:   for $j \in V_i$do
8:     RECV $f_j^{t-1}$ from learner $j$
9:   end for
10:  Perform one epoch with learning function in equation (8)
11: end for
```

Figure 1 shows a diagram of a parallel layout of RLNC. 1. The training data set—input parameter to algorithm—is resampled in N files by the master process and located in a Network File System (NFS), from which each file is read by the respective slave process. According to this, a one-to-one relationship between slave processes and files is established.

The training process is accomplished in a number of (fixed) iterations. In each iteration, a slave process sends its information $f_i(x')$ regarding the predictions on data set of neighbors $\forall x' \in D^j$. Then, the slave process receives information $f_j(x), j \in V_i$ from its neighbors regarding the predictions $\forall x \in D^i$ in the local data set obtained in the previous step, in order to update the weights and biases, according to the learning function (7).

Once the training process has been completed, each slave sends its predictions for each example x of the testing set to the master, in order to calculate the output ensemble, using the aggregation function $F(x)$ described in Eq. (1).

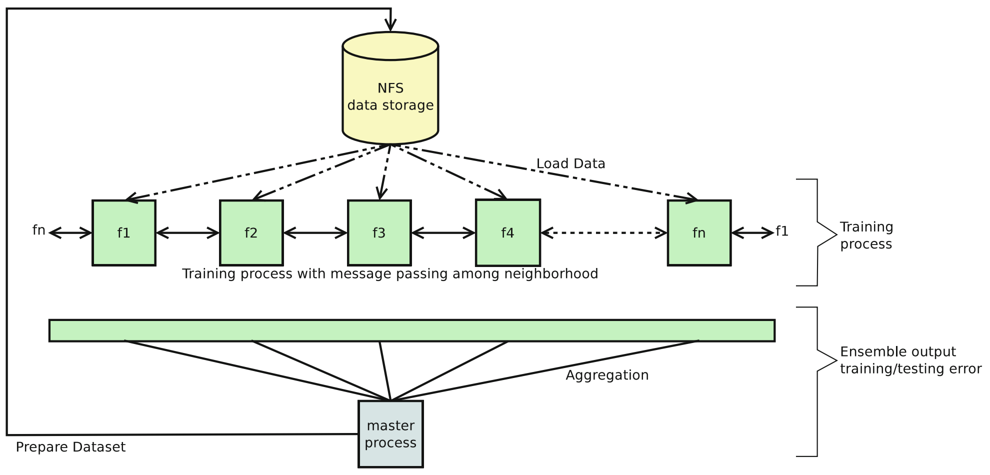
NFS
data storage
Load Data
fn
f1
f2
f3
f4
fn
f1
Training process with message passing among neighborhood
Training process
Ensemble output training/testing error
Aggregation
master process
Prepare Dataset

Fig. 1 Parallel processing of RLNC

4 Experimental results

In this section, we present an experimental section in which a comparison between own proposal virtual structure and the physical structure. A description of parameters and the environment software and hardware is shown. A parallel assessment that considers the scalability and efficiency both physical and virtual environment is made. The comparison between execution environments considers both the size of the problem and number of processors. Table 1 shows a brief description of the parameters.

The experiment results were obtained using the following configurations for hardware and software:

- Hardware: six computers with the following specifications:
  - Processor: Intel Core2 Quad CPU Q9400 @ 2.66GHz
  - Memory: 4 GB
  - Network Interconnect: Gigabit Ethernet
- Software:
  - Operating System: Centos 5.4 x86_64
  - Kernel Version: 2.6.18
  - Compiler: Intel icc 10.1
  - MPI Library: Mpich2 1.1.1
- NFS Server:
  - Processor: Intel Core2 CPU E7400 @ 2.8GHz
  - Memory: 3 GB
  - Network Interconnect: Gigabit Ethernet
  - Operating System: Ubuntu 9.10 x86_64
  - Kernel Version: 2.6.31

The configuration of the multicore cluster used in this work consists of six computation nodes that use Intel quadcore processors, each one has the same hardware and software versions, one NFS server and one GigaEthernet switch, where has been implemented a LAM/MPI cluster [20]. Virtualbox has been installed in each compute node. One virtual machine has been created using the same hardware and software versions used for the physical compute nodes.

The performance results of physical and virtual environment and a comparison between them are shown using Friedman data set [9], this analysis takes into account the performance metrics, speedup, and efficiency. The execution time for each experiment was obtained by an average of 20 runs. Two curves are shown for each environment in order to present the algorithm behavior when the number of

Table 1 Algorithm parameters

Parameter	Description	Values
p	Number of processors (cores)	{1, 2, 3, 4, 6, 8, 12, 16, 20, 24}
N	Number of machines in the ensemble	50 (fixed)
v	Neighborhood order	1 (fixed)
m	Size of the problem (number of inputs)	$\{250, 500, \ldots, 2,500\}$
Q	Number of iterations during training	100 (fixed)
η	Influence of neighborhood in training process	0.95 (fixed)
β	Number of executions of each experiment	10 (fixed)

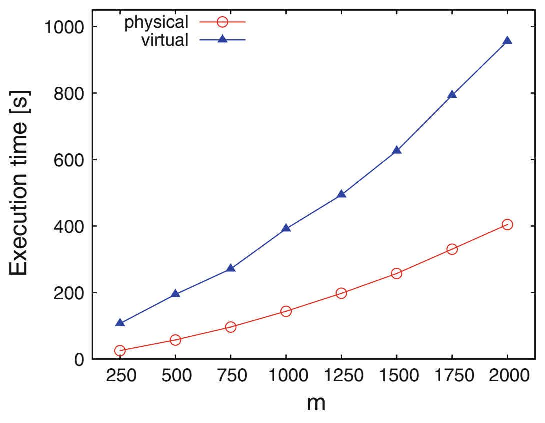

Fig. 2 Comparison of sequential execution time between both virtual and physical environments

processors (p) is increased, while the number of inputs (m) is fixed.

Figure 2 shows the empiric computational complexity, using one processor, obtained from both physical and virtual clusters. It shows that the overhead introduced by the virtualization widens as m grows, which is reflected as an increase in the differences between execution times.

Figure 3a and b show the behavior of the algorithm in terms of scalability. It can be observed that over 750 data inputs the algorithm begins to obtain good scalability and efficiency—over 80%—due to the increase in the number of processors, the efficiency decreases slowly; this is a characteristic behavior of parallel algorithms. Figure 3c and d show the variations of efficiency concerning the increase in the number of processors. The curves for $m \in \{250, 500\}$ fall rapidly, obtaining values of efficiency under 80%, it is observed that these curves tend to separate gradually one from the other.

Given the speedup reached by RLNC and according to [4], the algorithm has good scalability, so that it has the potential to explode the parallelism; it allows scaling the training process to a larger number of data inputs.

In this implementation $N > p$, each processor performs more than one MPI process. The node–core combination, used in the experiment, suggests that by this particular parallel approach, it can be obtained better results in terms of execution time, when the node–core combination gives more weight to the distribution in more nodes rather than the intensive use of all available cores in each node. An average result correspond to the balance between nodes and cores, for example, the use of 6 processors composed by 3 nodes using 2 cores in each node. This permits to obtain better execution time than the use of 2 nodes and 3 cores in each node. The best combination can be obtained

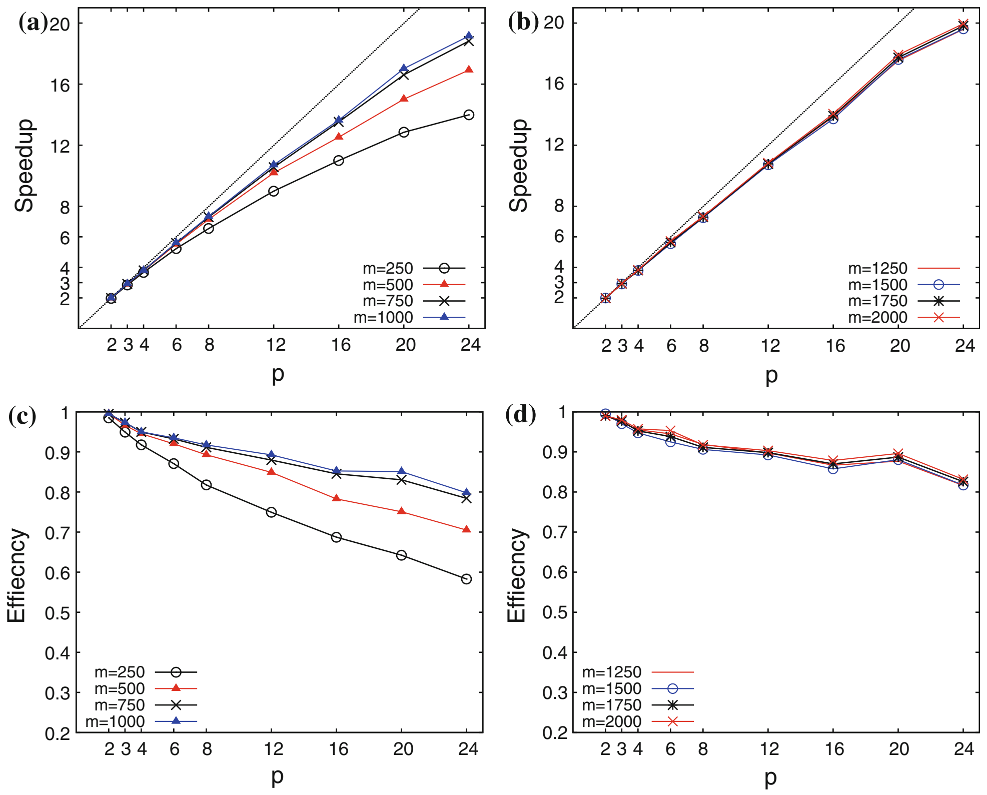

Fig. 3 Results obtained from parallel performance evaluation on physical cluster: the behavior of speedup v/s p with fixed values of m is shown in (**a**) and (**b**), where the *dotted line* denotes the linear speedup. The behavior of efficiency v/s p with values of m fixed is shown in (**c**) and (**d**)

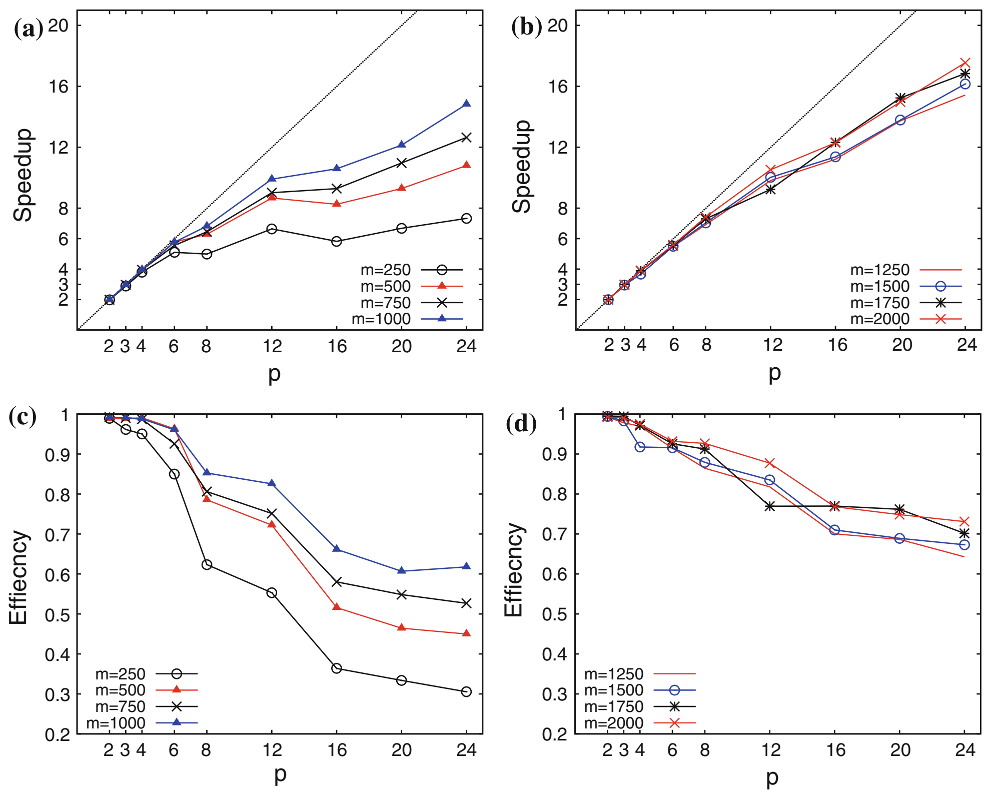

Fig. 4 Results obtained from parallel performance evaluation on virtual cluster: speedup v/s p in Figures (**a**) and (**b**), where the *dotted line* denotes the linear speedup. Figures (**c**) and (**d**) show efficiency v/s number of cores p

using 6 nodes with 1 core in each node. When $p \leq 12$, the best node–core combination was used, according to the selection criteria previously explained. In other cases, when $p = \{16, 20, 24\}$, all cores available per node are used, which introduces an additional overhead to the operating system scheduler.

Figure 4 shows the behavior in terms of scalability and variations of efficiency on virtual machine cluster. Figure 4a and b show the scalability, in which the overhead introduced by the use of all cores per node, when $p = \{16, 20, 24\}$, decreases its scaling potential.

Figure 4c and d show the variations of the efficiency in our virtual cluster environment. Considering the overhead introduced by virtualization, results are similar in most cases, when $p \leq 12$, showing an efficiency difference of the physical environment lower than 15%; except when $m = 250$; in this case, a poor efficiency in the virtual environment is observed.

In a physical environment using $m \geq 750$, the efficiency reaches values over 80% when $p \leq 12$; henceforth, the efficiency falls faster due to the overhead produced by the use of all cores available in each node when executing the parallel algorithm. This decreases the resources available for carrying out the tasks of the virtualized operating system.

The previously described overhead—with the 6 compute nodes cluster—can be reduced by including more compute nodes in the cluster, which allows, for instance, to achieve $p = 16$ cores by the use of a combination of 8 compute nodes and 2 cores per node. This occurs because RLNC performs better when the node–core combination leaves some cores available in each compute node.

5 Conclusions and future work

In this work, we have presented a testing environment generated by a full virtualization approach in which both parameter control and execution are carried out.

Comparable results are obtained between virtual and physical environments, taking into account the execution of the algorithm using a node–core configuration that allows having at least one core available for the virtual OS tasks. These node–core combinations have less overhead than those ones that use all cores on each node.

The results obtained in this research are associated with a specific type of virtualization and virtual machine monitor (VirtualBox); therefore, future work will be aimed to extend this research to other virtual environments, such as Xen and VMWare, a HPC cluster with better performance than the cluster environment available to the development of this research. This new cluster allows us to obtain results from a larger amount of data inputs. In addition, it would be interesting to study the way in which virtualization allows the construction of a flexible cluster testing environment adaptable to the ensemble structure. This study would allow us to perform research such as behavior analysis, parameter control, cross-validation, and testing new approaches of ensembles of neural networks.

Acknowledgments This work was supported by the following Research Grants: Fondecyt 1070220 , 1110854, and FB0821 Centro Cientìfico Tecnológico de Valparaíso. Partial support was also received from CONICYT (Chile) Ph.D. Grant 21080414, MECESUP Ph.D. Grant FSM0707-D3044, and project FB0821-FB_15LS_10.

References

1. Anastasiadis A, Magoulas G (2006) Analysing the localisation sites of proteins through neural networks ensembles. Neural Comput Appl 15:277–288. doi:10.1007/s00521-006-0029-y
2. Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A (2003) Xen and the art of virtualization. In: SOSP '03: proceedings of the nineteenth ACM symposium on operating systems principles. ACM, New York, pp 164–177
3. Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140
4. Breshears C (2009) The art of concurrency: a thread monkey's guide to writing parallel applications. O'Reilly Media, California
5. Brown G (2004) Diversity in neural network ensembles. PhD thesis, School of Computer Science, University of Birmingham
6. Chai L, Gao Q, Panda DK (2007) Understanding the impact of multicore architecture in cluster computing: A case study with intel dual-core system. In: Cluster computing and the grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on, pp 471–478
7. Crawford IL, Wadleigh KR (2000) Software optimization for high performance computers. Prentice Hall PTR, Upper Saddle River
8. Crosby S, Brown D (2007) The virtualization reality. Queue 4(10):34–41
9. Friedman JH (1991) Multivariate adaptive regression splines
10. Gepner P, Kowalik MF (2006) Multicore processors: new way to achieve high system performance. In: PARELEC '06: proceedings of the international symposium on parallel computing in electrical engineering. IEEE Computer Society, Washington, DC, USA, pp 9–13
11. Grandvalet Y (2004) Bagging equalizes influence. Mach Learn 55(3):251–270
12. Gropp W, Lusk E, Skjellum A (1999) Using MPI: portable parallel programming with the message passing interface. MIT Press, Cambridge
13. Hiroaki I, Edahiro M, Sakai J (2007) Towards scalable and secure execution platform for embedded systems. In: ASP-DAC '07: proceedings of the 2007 Asia and South pacific design automation conference. IEEE Computer Society, Washington, DC, USA, pp 350–354
14. Huang W, Liu J, Abali B, Panda DK (2006) A case for high performance computing with virtual machines. In: ICS '06: proceedings of the 20th annual international conference on supercomputing. ACM, New York, NY, USA, pp 125–134
15. Liu B, Cui Q, Jiang T, Ma S (2004) A combinational feature selection and ensemble neural network method for classification of gene expression data. BMC Bioinformatics 5:136
16. Maqsood I, Khan MR, Abraham A (2004) An ensemble of neural networks for weather forecasting. Neural Comput Appl 13:112–122. doi:10.1007/s00521-004-0413-4
17. Ñanculef R, Valle C, Allende H, Moraga C (2006) Ensemble learning with local diversity. In: ICANN (1), pp 264–273
18. Ñanculef R, Valle C, Allende H, Moraga C (2006) Local negative correlation with resampling. In: IDEAL, pp 570–577
19. Necaise RD (2001) Using vmware for dual operating systems. J Comput Small Coll 17(2):294–300
20. Ong H, Farrell PA (2000) Performance comparison of lam/mpi, mpich, and mvich on a linux cluster connected by a gigabit ethernet network. In: ALS'00: proceedings of the 4th annual Linux Showcase & Conference. USENIX Association, Berkeley, CA, USA, pp 31–31
21. Ranadive A, Kesavan M, Gavrilovska A, Schwan K (2008) Performance implications of virtualizing multicore cluster machines. In: HPCVirt '08: proceedings of the 2nd workshop on System-level virtualization for high performance computing. ACM, New York, NY, USA, pp 1–8
22. Valle C, Saravia F, Allende H, Monge R, Fernández C (2010) Parallel approach for ensemble learning with locally coupled neural networks. Neural Process Lett, 32:9157:277–9157:291
23. Watson J (2008) Virtualbox: bits and bytes masquerading as machines. Linux J 2008(166):1
24. Wells PM, Chakraborty K, Sohi GS (2009) Dynamic heterogeneity and the need for multicore virtualization. SIGOPS Oper Syst Rev 43(2):5–14
25. Whitaker A, Shaw M, Gribble SD (2002) Denali: lightweight virtual machines for distributed and networked applications. In: Proceedings of the USENIX Annual Technical Conference
26. Zhang PG (2007) A neural network ensemble method with jittered training data for time series forecasting. Inf Sci 177(23): 5329–5346