A Parallel Implementation of the K Nearest Neighbours Classifier in Three Levels: Threads, MPI Processes and the Grid [⋆]

G. Aparício, I. Blanquer, V. Hernández

Instituto de las Aplicaciones de las Tecnologías de la Información y Comunicaciones Avanzadas - ITACA
Universidad Politécnica de Valencia. Camino de Vera s/n 46022 Valencia, Spain
`{ gaparicio, iblanque, vhernand }`@*itaca.upv.es*
Tel. +34963877356, Fax +34963877359

Abstract. The work described in this paper tackles the problem of data mining and classification of large amounts of data using the K nearest neighbours classifier (KNN) [1]. The large computing demand of this process is solved with a parallel computing implementation specially designed to work in Grid environments of multiprocessor computer farms. The different parallel computing approaches (intra-node, inter-node and inter-organisations) are not sufficient by themselves to face the computing demand of such a big problem. Instead of using parallel techniques separately, we propose to combine three of them considering the parallelism grain of the different parts of the problem. The main purpose is to complete a 1 month-CPU job in a few hours. The technologies that are being used are the EGEE Grid Computing Infrastructure running the Large Hadron Collider Computing Grid (LCG 2.6) middleware [2], MPI [3] and POSIX [4] threads. Finally, we compare the results obtained with the most popular and used tools to understand the importance of this strategy.

Topics. Grid, Parallel Computing, Threads and Data Mining.

1 Introduction

Data mining is a recently created concept that groups different techniques of data analysis and model extraction. The main purpose is the extraction of hidden predictive information from large databases. In this way, data mining is a helpful technology to get profit of the great amount of unused generated data. Our interest is focus on classification (automatic choice of the category in which a piece of information will more likely fall into). The target of this work is the development of a set of tools to assist large-scale epidemiology studies. Thus the initial hypothesis is a large database with registers in which most of them

[⋆] The authors wish to thank the financial support received from the Spanish Ministry of Science and Technology to develop the GRID-IT project (TIC2003-01318) are labelled and in which the process will predict the label for a few of them. Considering this situation, we select the K nearest neighbours method [1] as the most suitable classification method to obtain the needed results, i.e., the predicted labels. The work will concentrate on speeding up the performance. The analysis of the accuracy and goodness of the predictions are not in the scope of this work.

2 K nearest neighbours

One of the most popular instance-based classification techniques is the K Nearest Neighbours method [1]. This is a generalisation of the one nearest neighbour rule. This method is appropriate when dealing with a large set of labelled registers or instances and a small group of non-labelled registers to be classified with the most probable label. Our hypothesis is that we work on data where there is an strong relationship between the multidimensional distance of each entry with the rest and the label of the instances. In this way, the 1-NN (one nearest neighbour [1]) consists on assigning to the non-labelled registers, the label of the nearest labelled register. Based on that, Cover & Hart [1] implemented a variant known as K-NN (K Nearest Neighbours) that applies the same philosophy that 1-NN but considering the most frequent label in the K Nearest Neighbours instead. The K parameter is very important and the optimal value will depend on many factors, such as the data nature, and it has to be chosen experimentally.

The cross validation test is the most popular test technique used. Starting from a set of labelled registers, those are divided into B blocks. One block will be used as the test set and the others as the training set. This operation is performed with the B blocks. The K-NN classifier is applied to any register in the test block and compares the label assigned with the one previously defined. If the labels differ, an error is recorded, and the consolidation of all the errors obtained for the whole test block is the validation error. Repeating this process by changing the test set to the rest of blocks, the sum of all the errors will be an approximation to the real error of the K-NN classifier with the chosen K parameter

$$err = \sum_{b=1}^{b=B} (err_b) \text{ where b is the chosen block.}$$

3 Three-layer Parallelism Architecture

The decision of using the three layer parallelism is due to the difficulty of managing a large amount of data (especially when dealing epidemiology databases containing several million registers) and the poor results of traditional sequential approaches. Our purpose is performing a set of experiments with different values of K in the K-NN method to determine the optimal value for this K according to the lowest error in a cross validation test. Once the optimal value for K is determined, we will be able to classify the non-labelled registers with the K-NN method. The main aim of the work will be on reducing computing time especially for epidemiology analysis and support.

The evaluation of the error for each value of K takes around 18 CPU hours using a state-of-the-art computer and a database of 1 million records and 20 fields per record. Thus, a complete optimisation process of 10 different values of K would take more than 7 CPU days, which would be unmanageable in a production environment. However, the process is intrinsically parallel, presenting two clear levels of parallelism. In a first level, each evaluation of the error using different values of K is totally independent, since it consists on computing the whole classification and cross-validation process for each value of K, using the same input database. This process consists on computing the K minimal distances to all registers and selecting a block to act as the test set. This block will be re-labelled using the rest of the database as the training set. Labels are assigned considering the labels of the K Nearest Neighbours. This process will be repeated selecting different blocks of the database as training sets in order to cover the whole database. Each validation using a different block is independent. However, the computational cost of this process is on the order of cN^2, being c the number of flops for computing a single distance between two registers and N the number of registers in the database (directly affecting communication cost). Thus, a trade-off solution must be applied to obtain the maximum performance considering the best granularity.

In the frame of this scenario, three parallelism approaches can be considered. The conditions of each one concerning the problem of this article are the following:

- Grid Computing. This technique implies the coarsest granularity. Grid Computing deals with the concurrent usage of different computing resources in different administrative domains in a similar approach as a large-scale batch queue. Inter-resource communication is not usually available due to the long latencies, the internal configuration of nodes in resource providers and the overhead of the security policies. Data access thus is mainly performed through the job submission process and shared repositories, using ftp-like protocols. In this paradigm, the minimal running entity is the job, interacting with the rest of the jobs through input and output files.
- Message-Passing Parallel Computing. This technique is proven to be very efficient in most medium-grain problems in which communication costs are on an order of magnitude lower than computing cost. Typically, jobs are fairly symmetrical and run on homogeneous nodes connected through a fast network. Security policies are not applied in communication and data exchange is performed through message passing.
- Shared-Memory Parallel Computing. This constitutes the finest-grain parallelism. Applicable in very coupled and homogeneous environments, different threads concurrently execute a common program on different fragments of data. Data is exchanged through shared regions and contention mechanisms.

Generally the scaling factor of those systems is low, due to hardware constraints and speed-ups are good.

Our proposal is to combine the three levels to achieve the maximum performance. Using shared-memory approaches only would lead to small speed-ups (limited by the number of available processors) and the need of shared-memory supercomputers. The combination of distributed-memory and shared-memory approaches would increase notably the speed-up, since computing farms can reach without performance losses many tens of bi-processor nodes. However, and considering that our problem is massively parallel, more powerful configurations could be efficiently used. Thus, the coordinated use of several computing farms is a reasonable choice considering the availability of those systems. In this case, grid computing constitutes an efficient way to organise and manage different computing resources in different administrative domains. So, in order to achieve the maximum performance, we have decided to combine three different techniques of parallel computing: GRID technology, MPI programming and POSIX threads. Considering the different characteristics of each approach, the problem of classification must be structured to obtain the maximum efficiency from each one. According to this, we have chosen the EGEE infrastructure currently running the LCG 2.7 Grid Middleware [2]. Command Line Interface (CLI) will be used to implement the scripts and the programmes for submitting several experiments with different values for K. Each experiment is performed concurrently by several MPI processes to divide the cross validation into a simple test-training partition validation. Finally different sub-blocks of each MPI process testing partition are computed on the different POSIX threads created in a MPI process and executed within the processors of a node.

An example of a tree diagram of our approach is printed in figure 1. In this figure we can see the evolution of the work. It begins at the root of the tree with the submission of K different LCG Grid jobs (where the value of K will determine for each LCG job the number of neighbours to which the distances are computed). In a second phase, MPI parallel process are executed. We choose the same number of MPI processes as cross validation blocks, although other factors of parallelism grain could be analysed in the future. In the third phase, MPI processes are split into threads, according to the features of the target hardware resources and the experimental results.

4 Implementation

The implementation of the three-layer model mainly considers three components:

– Parallel KNN module. This component implements the KNN classification and cross-validation algorithm using MPI and POSIX Threads. It is an autonomous executable that takes as input the reference to the labelled registers file name, the reference to the file that contains the registers we want to

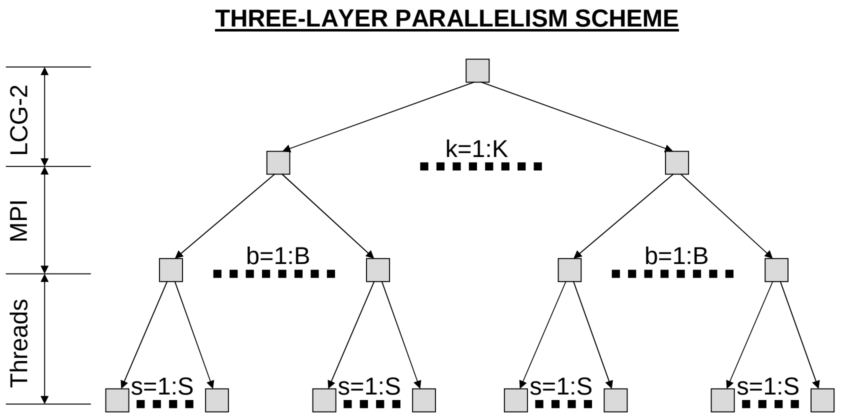


Fig. 1. Three-layer parallelism scheme.

label and the K value and produces a different output file depending on our demands, being possible to show the labels assigned to the target registers or to obtain a statistical summarized file.

- Grid scripts. They implement the selection of the rightmost computing resources, the job description file, the start-up script for the parallel executable, the job submission and monitoring and the job output retrieval. All these tasks are implemented through scripts that make use of the CLI.
- Java Interface. It implements a user-friendly interface to select the data and the parameters for the Grid jobs and to retrieve their output.

The parallel KNN module comprises the MPI and POSIX Threads Computing levels. The synchronised execution of different instances of this processing module is performed through the Grid scripts.

The figure 2 shows us the global process, according to a chronological view. In the upper part of the diagram we can see subprocesses classified by functionality and in the lower part they are classified by technology.

4.1 Grid Computing Level

One typical application of Grid technology is multi-parametrical runs. The difficulty in establishing efficient communications among independent submitted jobs in a Grid environment has been traditionally an important barrier. In our case, different experiments with different values of K in the K-NN method, constitute clearly a multi-parametrical task which can be achieved by different LCG Grid jobs.

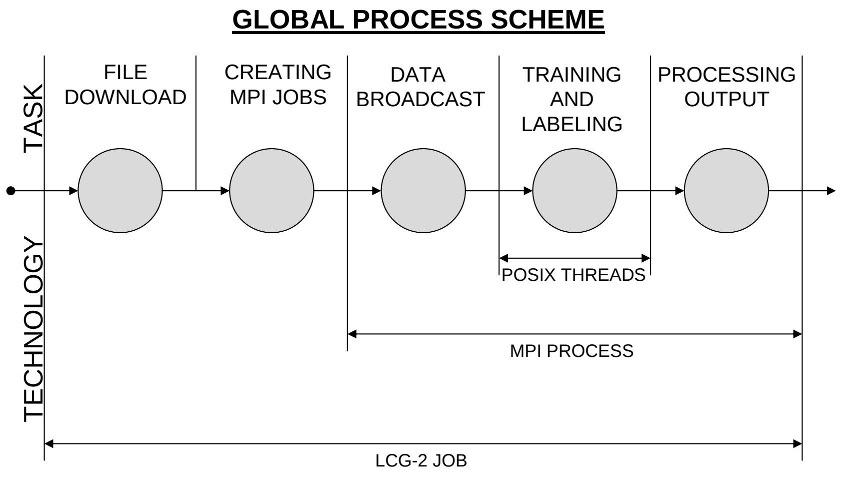


Fig. 2. Global process scheme.

The Grid infrastructure selected is EGEE (Enabling Grids in E-sciencE). This is the largest production infrastructure available for research world-wide, integrating, in April 2006, more than 35000 computers and more than 3 Petabytes of storage in 180 sites. This infrastructure runs currently the LCG 2.7 middleware, although it will migrate to gLite 3.0 during Summer 2006. gLite 3.0 and LCG 2.7 share a major part of components, so migration is feasible. Both LCG 2.7 and gLite 3.0 are batch-oriented Grid middlewares and consider the same computing structure, which comprises the following components:

– Computing Resources. The computing resources are organised in the form of Computing Elements (CEs), and Working Nodes (WNs). CEs are the front-ends and visible entry-points to computational farms of WNs. CEs implement the necessary batch queues to manage the jobs in the WNs and keep track of the status of both jobs and resources. CEs by themselves do not usually run any job. WNs typically can communicate within the cluster and although inbound connections are allowed, direct communication within WNs of different resources is not possible. From the users' point of view, the resource is the CE.
– Storage Resources. The files in the EGEE infrastructure are stored in a distributed way in many Storage Elements (SEs). Each CE should have a proximum SE, which could be shared between CEs. The data stored in the SEs is organised through the catalogue system.
– Workload Management. The destination of a job (a queue in a CE) can be directly selected by the user, although the more effective way is to rely on the Workload Management System. This system queries the status of all the CEs associated to it and decides the most suitable one considering the job and requirements, the site features and the site workload. gLite and

LCG have different (but interoperable) systems for job management. LCG comprises the Resource Broker (RB) service and gLite the Workload Management System (WMS), which implements larger functionality and provides better performance.

– Storage Catalogue. Data stored on the SEs is organised through Storage Catalogues. Storage Catalogues keep track of the Logical File Names (LFN) of the files stored. The LFN is a unique logical identifier of a file or group of replicated files. The Storage Catalogue has the information of the real location in the storage system of all the replicas of the file pointed out by an LFN. Metadata of the file is stored in other special catalogue systems (such as AMGA).
– User Management. Users are organised in Virtual Organisations (VOs). Typically, users in a VO have the same authorisation rights to access the resources. This reduces the burden of managing individual security policies. VOs publish lists of authorised users through their digital certificates and Distinguished Names, which are locally copied in all the resources. More fine-grain policies are implemented through other systems such as VOMS (Virtual Organisation Management System), which is also provided in the new release of gLite 3.0.
– System Information. The information of the system (status of the jobs and resources mainly) is published in a hierarchical model by the sites and the monitoring system. Each information node in EGEE provides information about the resources registered to it. Site information points have information about the sites' resources. Federation information systems register information about the resources of a whole federation, up to the central information system.

In order to execute a job in the LCG environment, a Job Description File must be written according to the Job Description Language (JDL). This file defines the input and output files, the executable file and the running parameters and the program requirements. The executable is typically a shell script that copies all necessary data locally on the resource and performs other preliminary steps (such as re-compiling, executable permissions, etc.). Job is submitted through the specific commands or API calls and enters in a cycle of states (submitted - waiting - ready - scheduled - running - done - outputready - cleared).

Once the LCG job is assigned to a Computing Element, a shell-script is executed. The shell-script initial instructions will fetch the databases stored in the Grid, including both training registers and labelling registers, since not only training but also classification is performed in the last phase of the job. This approach will reduce the Grid waiting time. When all the necessary data are downloaded on the computing nodes, the classification process can begin. This is achieved using an MPI process that will be the responsible of making a test with an assigned block. The sum of all the block test errors will be the cross validation error, i.e., the information we are requesting to decide the optimal K value and then use it to classify the non-labelled registers.

The tasks that must be implemented for delivering the above functionality are:

– Selection of the rightmost Computing Resource. Resources in the EGEE Grid are accessed through Workload Management Systems, such as the LCG Resource Broker. Those resources are selected according to the job features and the VO policies. In our case, the main requirement was the support of MPI. The resources were ranked considering their proximity to Storage Resources where replicas of the database are stored and other performance criteria (mainly the number of free CPUs, historical average length of the submission queue). A 'black list' of faulty resources is kept to exclude those resources which produced execution errors in the past.
– Submission and Resubmission. Once the resources are identified, input data for each job is packed and jobs are submitted along with all the needed information (including references to the stored databases). The status of the job is periodically monitored and jobs are resubmitted to a new computing resource if scheduling time exceeds a predefined threshold. Aborted jobs are resubmitted up to a predefined number of times. The submission of a job implies, as a previous step, compiling and linking the executable with the MPI libraries available in each computing resource. LCG nodes supporting MPI are configured to export the location of MPI libraries and header files.
– Monitoring and Output Retrieval. Jobs being executed are monitored through the corresponding scripts. Once finished, output data are retrieved and user is notified via e-mail (obtained automatically from the Distinguished Name of the certificate or given as a parameter).

Final result of all the process is the cross-validation error for the execution of KNN for a specific value of K. Results are presented as available and sorted by the magnitude of the error.

4.2 MPI and POSIX Threads Computing Level

The MPI executable is an autonomous programme that computes the distance evaluation, cross-validation and labelling of the registers of a database.

MPI process 0 will be the responsible to load and broadcast databases to the rest of MPI processes that run in other nodes of the cluster selected in the Grid Infrastructure for each job. Databases are replicated among all the processors. The computation of the distance requires considering all the registers each time. Other distributions could be considered if memory is insufficient, although they will require additional communication cost, since they must involve intermediate data exchange among processes. The distribution of large amounts of data does not imply an important handicap since the communication is performed inside the cluster farm and not through the Grid.

MPI process 0 also normalises the database to ensure that all fields of each register are considered with the same weight. Registers are evenly distributed among the processors and within each processor evenly among the threads. A list of the K nearest registers is updated during the process. This process is run for the block of registers selected as the test set in each computer. The errors are computed as the number of wrongly assigned labels. Finally, labels are assigned at the end of the process to reduce the overhead of redistributing the data again.

Each block of test-set registers considered in a processor is processed by different threads. POSIX threads are created in each MPI process dealing with the distance computation and labelling of a portion of the testing set. The use of POSIX threads permits exploiting efficiently the multiprocessor capability that modern clusters have. Moreover, the process do not imply conflicts neither on write access to common variables nor on synchronisation. Experiments also prove that the consideration of the hyper-threading capabilities of current processors provide an additional gain factor in the speed-up, being able to run more threads than physical processors are available.

The moderated cost of this process (in the quadratic order) and the large amount of data to be exchanged makes this problem suitable for parallel computing rather than Grid computing. Complexity can be increased by considering more costly computations of the distances (currently a homogeneous Euclidean distance) considering different weights or distance metrics for different fields, or considering more complex error metrics, such as distances to the right label.

4.3 User Interface

In order to ease the process of creating experiments, submitting the jobs and monitoring the results, a java-based interface has been implemented. This interface co-ordinately execute the necessary scripts to deal with the Grid job submission, monitoring and output retrieval.

The interface enables an authenticated and authorised user to log in the system and upload the test and training sets on a specific SE. Target CEs will be selected according to the availability of computing resources, the support of MPI and the proximity to a SE publishing a replica of the databases.

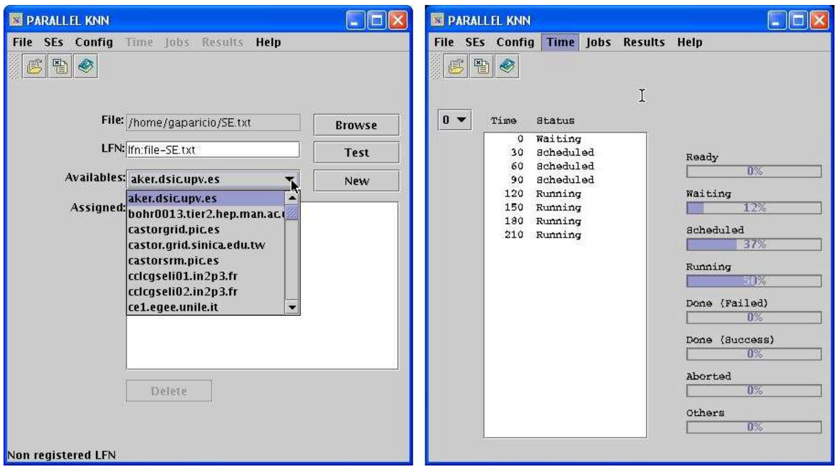
Then, jobs are automatically constructed considering the range of values of K that will be screened. Jobs are submitted through the interface and their status is monitored either at global level (percentage of jobs in each state) or individually by jobs. Jobs are automatically submitted if an error is produced (although a maximum retry count is reached) or if they keep on waiting on a queue for an excessive time, choosing in both cases a different computing resource.

Finally, the result of the jobs can be dynamically consulted. The user is notified when a job has finished through e-mail.

Figure 3 shows a couple of snapshots of the application interface.

5 Results

We have done different experiments in three scenarios. First, using the command-line WEKA [5] java K-NN class (3.4.5 release, a very well-known and widely used classification tool); second, using a sequential tool we have implemented in


PARALLEL KNN
File SEs Config Time Jobs Results Help
Browse
Test
New
Delete
PARALLEL KNN
File SEs Config Time Jobs Results Help

```
File: /home/gaparicio/SE.txt
LFN: lfn:file-SE.txt
Availables: aker.dsic.upv.es
  aker.dsic.upv.es
  bohr0013.tier2.hep.man.ac.
  castorgrid.pic.es
  castor.grid.sinica.edu.tw
  castorsrm.pic.es
  cclcgseli01.in2p3.fr
  cclcgseli02.in2p3.fr
  ce1.egee.unile.it
Assigned:
Non registered LFN
```

```
0   Time  Status
       0  Waiting
      30  Scheduled
      60  Scheduled
      90  Scheduled
     120  Running
     150  Running
     180  Running
     210  Running

Ready            0%
Waiting          12%
Scheduled        37%
Running          50%
Done (Failed)    0%
Done (Success)   0%
Aborted          0%
Others           0%
```

Fig. 3. Snapshots of the user interface application.

”C”; and, third, by using the three-layer parallel K-NN tool. We have used two different training databases, one with one hundred thousand labelled registers and another one with one million labelled registers. The number of fields of the two databases was 20, plus the label field.

The performance of the version implemented in ”C” language was very efficient comparing to the results with WEKA, mainly due to the fact that WEKA is implemented in Java. A linear speed-up is obtained using grid technology since the experiments are independent. The gain in the MPI parallelisation approach has been above a factor of 9.5 with 10 biprocessors nodes (PIII Xeon 3GHz. on an SCI 3D torus network) and the gain using four threads in each node is above an additional 1.5 factor. We selected four threads considering that each node has two hyper-threading processors. This gain is not as linear as in the other cases since not all the process has been implemented using threads. Thus, the gain of using the MPI and threads parallel technologies is above 15, and the total advantage from WEKA multiplies by 6. Moreover, as it was mentioned before, Grid scales linearly. Figure 4 summarizes the results obtained.

This figure reflects the Speed Up comparing to WEKA to the sequential process, the MPI parallel process and the three-layer parallelism process.

6 Conclusion

The results of our work have been very encouraging. The approach based on a three-layer parallelism is a very effective way to get the best performances and give us a hopeful vision of data mining in the Grid. The classification of the registers, including the identification of the optimal K value in the K-NN method on a database of one million registers took less than 6 hours, i.e., a single nightly

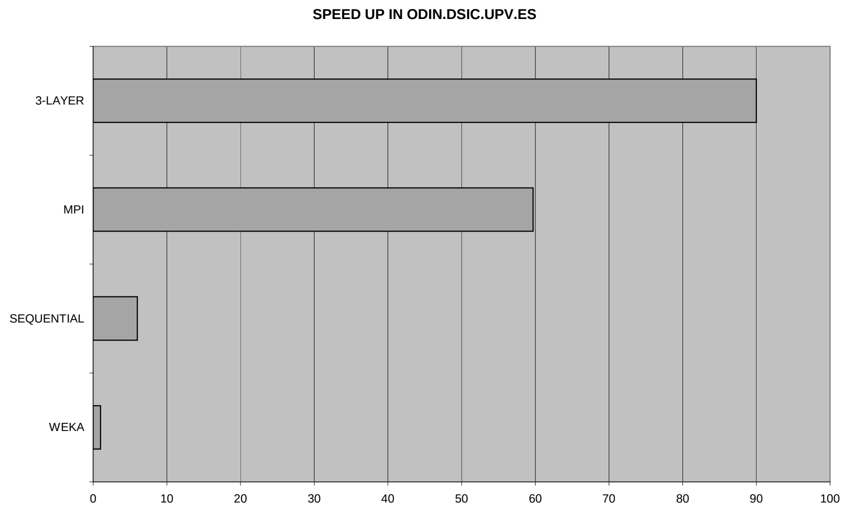


Fig. 4. Speed Up results.

run. For a comparison, this results implies an speed-up larger than 90 compared to the equivalent WEKA K-NN sequential execution, i.e., more than a month waiting time. Considering that our efforts are routed to biomedical prediction, this advantage would enable, for example, classifying the information recorded in Primary Care each day (in the order of millions of records) and its automatic classification.

References

1. T.M.Cover, P.E.Hart: Nearest neighbour pattern recognition. IEEE Trans. on Information Theory **13(1)** (1967) 2127
2. LCG: Large hadron collider computing grid. Technical report (2005)
3. MPIForum: Message Passing Interface Standard. (2005)
4. RedHat: The Native POSIX Thread Library for Linux. (2005)
5. E. Frank, M. Hall, L.T.: Weka 3: Data Mining Software in Java. (2005)