Multi-Class AdaBoost Learning of Facial Feature Selection through Grid Computing

Mian Zhou[*][†][¶], Hong Wei[†], Ian Bland[†], Anthony Worrall[†], David Spence[‡], Xiangjun Wang[§], Pengcheng Wen[§] and Feng Liu[§]

[*]State Key Laboratory of Precision Measuring Technology and Instruments, Tianjin University, Tianjin, China, 300204
email: zhoumian@hotmail.com
[†]School of Systems Engineering, University of Reading, Reading, United Kingdom, RG6 6AY
[‡]IT Service, University of Reading, Reading, United Kingdom, RG6 6AY
[§]College of Precision Instrument and Opto-Electronics Engineering, Tianjin University, Tianjin, China, 300072
[¶]School of Educational Technology and Information Services, Tianjin Foreign Studies University, Tianjin, China, 300206

***Abstract*—AdaBoost is an efficient method for producing a highly accurate learning algorithm by assembling multiple classifiers, but it is also widely known for its long duration of off-line learning. Especially, when it is applied for feature selection for object detection, its learning process is to exhaustively evaluate every feature in a large set. With the increasing of image resolution and complexity of feature transformation approaches, the computational time will be extremely long, which makes the large scale AdaBoost learning very difficult. In this paper, we have employed Grid Computing to solve the difficulty. The proposed algorithm is to select the most significant features for face recognition. The selection algorithm is derived from multi-class AdaBoost, which exhaustively evaluate every feature from a large set. The deployed Grid Computing system is actually used for High Throughput Computing specialised on advanced resource management. To utilizing Grid Computing on the feature selection process, we have improved multi-class AdaBoost learning algorithm with parallel structure, so that the task of High Performance Computing is accomplished in the environment of High Throughput Computing. With Grid Computing, selecting 200 features from a large set of 30240 features is finished in 20 days, while without Grid Computing the time would be more than two years. It shows that Grid Computing brings vast advantage to computer vision, machine learning, image processing, and pattern recognition.**

I. Introduction

In recent years, with the development of more precise image acquisition equipment and more advanced learning algorithm, machine vision become highly computational intensive. In some cases, processing data is obtained with high resolution, multiple frames, various dimensionality, different scales, etc. It leads to massive computational demand for modern machine vision application. AdaBoost algorithm used for feature selection is just the case. To find significant features representing human faces, Viola and Jones [1] employ AdaBoost to select 200 rectangle features from over 180000 features. The detection operates rapidly in real-time, however the training procedure is notorious too long. Although some efforts [2] are made by simplifying processes AdaBoost learning in return for degrading performance, it still requires large amount of computational time without sacrificing accuracy, In this paper, we present a Grid Computing [3] solution for AdaBoost learning which utilises all computers over a local network within an organisation. A whole learning task is divided into hundreds of jobs, so that the jobs are processed simultaneously on individual computer. Finally the task can be accomplished in a limited time.

The remainder of the paper is organised as follows. Section II introduces our grid computing system. Section III gives an overview of the algorithm used in multi-class AdaBoost, and Section IV describes the implementation of the grid computing. Section V gives a conclusion on grid computing in machine vision application.

II. Condor Grid Computing

A. Condor

The grid computing system used in this paper is *Condor*, which first went to stage in 1987 installed as a High Throughput Computing system at the University of Wisconsin-Madison. After more than 20 years growing, it has been expended into a huge system consisting of over 1000 workstations and cluster CPUs at the same site. Also, it becomes a popular Grid computing solution serving dozens of real users from hundreds of organisations in industry, government, and academia. By February 2005, Condor has been deployed on at least 60428 CPUs in 1265 organisations in 39 distinct countries, and the largest Condor seen so far consists 5608 CPUs. [4]

From infrastructure of Condor, it is a specialised heterogeneous, autonomous, semi-available computing resource management system for compute-intensive jobs. The fully featured resource management system provides job management, scheduling policy, priority scheme, resource monitoring, and management. The Condor provides more power than traditional batch queueing system, which does not only allow high throughput computing but opportunistic computing. High throughput computing is referred as effectively utilisation of all available resources within a network to achieve large amounts of fault-tolerant computational power over a long-time scale, and opportunistic computing is to seek and utilise some resources whenever they are available.

There are three key mechanisms of Condor: *ClassAd*, *Job checkpoint & migration*, and *CPU scavenge*. The ClassAd provides a framework for matching jobs with resources available with user defined job requirements and job preferences. In practice, users normally define software and hardware configuration which are minimal requirements for running jobs. The job checkpoint & migration enables transparently recording a checkpoint on a computer and subsequently resume or migrate computation from the checkpoint file. Condor also adopts CPU scavenge mechanism to search and manage wasted CPU power from idle nodes across computer clusters. Typically the mechanism is used to compute effectively and prevent resources from wasting at night, during lunch, or even in scattered hours throughout the day when nodes are idle.

B. Reading Campus Grid

The University of Reading provides Grid computing service for scientific research on brain computer interfaces, weather forecast, climate change, DNA analysis and machine vision computing. The grid is based on a Condor pool consisting of about 300 machines. It is deployed by IT Services and the School of Systems Engineering.

These machines are located in general computer labs around the campus and the library for teaching purpose, so that they can be utilised for scientific computation during school vacations or at nights. There are three types of nodes in the Campus Grid: Submit Node, Master Node and Compute nodes. SN refers to the machine which users make use of to set off jobs. Master node is a management centre of all jobs in the Grid. In addition, a master node contains Condor pool which is used for queuing, scheduling, and prioritizing jobs. Compute nodes are machines which actually process the Campus Grid jobs. Normally, one machine is used for Submit node, another machine is used for master node, the rest of machines in the Campus are served as compute nodes. In some cases, the master node and the submit node can be one machine, since submitting and managing jobs does not take much CPU resource. Figure 1 shows these three different types of nodes, and the path which jobs are going through the Grid. Users log into the submit node, submit jobs to the master node in the Linux-based command shell. The master node sets up a queue from the received jobs. When a compute node is available, *i.e.*, idle for a while, the master node will assign a job to it. When the node is occupied, the master node will stop the job, withdraw it to the Condor pool.

As it stands up to about 300 compute nodes (called node for short) could be available for running jobs. The actual number of nodes will rise and fall during the day as students occupy the machines, but overnight nearly all are available. Of the nodes approximately 15% have 2GB of memory available, 74% have 1GB, and 11% have 512MB. When a user logs on to a node in person then the node becomes no longer available to the Campus Grid. If there is a Grid job running, then it will be stopped. After a while, Condor will try restarting the stopped job on another node by using Checkpoint. To access the Campus Grid, a user needs to have an account and logs on

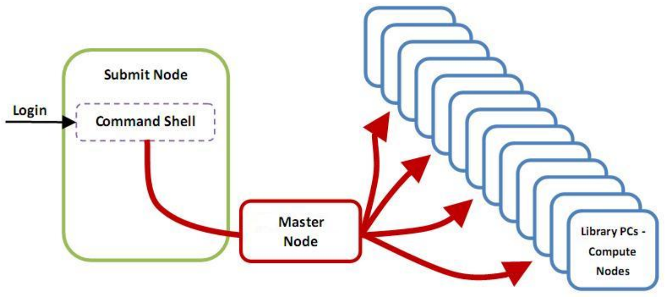

Fig. 1. Three types of nodes and path of jobs in the Grid

the submit node. Access is though Secure Shell (SSH) network protocol so that users communicate the Campus Grid using a secure channel between local computers and the submit node. All the nodes in the Campus Grid have access to Network File System (NFS) drive belonging to user's account. It avoids file transfer operation by users over the Grid, and the file access is transparent to users. Whatever a job creates, changes or removes a file on a node, it will be visible immediately on other nodes. The advantage is for saving results, since result from different node is saved in an unique space rather than different spaces for different nodes. The quota of NFS drive for each account is 10GB. If it is not enough, each node provides a shared space of 20GB which is used for temporary files. However, after jobs are finished, the temporary files are required to be deleted, and these files is invisible from other nodes.

Most 300 compute nodes generally run MS Windows locally, since they are deployed for all students for general purposes. However, the appropriate environment for Condor computing is Unix/Linux in the Campus. Hence, Cooperative Linux (CoLinux) [5] technology is used to tackle the problem. CoLinux is a port of the Linux kernel that allows it to run together with another operating system. In the campus, CoLinux is stored as a Linux image file on MS Windows computers. Once an MS Windows node is idle for 30 minutes, *i.e.* no I/O access, no keyboard stroke or no mouse input, the CoLinux image will be loaded into the memory and deployed as a standard MS Windows service for accepting Condor jobs. The CoLinux balances the usage between general computing purpose and Condor computing purpose. Also it improves the usability of Condor grid, so that users avoid dual-booting computers or adopting expensive visualisation software.

The Condor has two runtime environments called universes. One is *standard* universe, and the other is *vanilla* universe. The standard universe provides the mechanism to record a checkpoint and migrate a partially completed job to a new node. To use the standard universe, it is necessary to compile program with the special compiler provided by Condor. The vanilla universe provides a way to run jobs which do not require to re-link the Condor compiler. There is no job checkpoint or migrate mechanism under the vanilla universe. For access to files, jobs must either use a shared file system, or use Condor's File Transfer mechanism. Although the vanilla universe has many constraints on managing the jobs, the program is easily built by common compilers, such as **GNU GCC**.

III. Multi-class AdaBoost Learning of Feature Selection

The learning algorithm in this paper is used to perform feature selection for face recognition. The procedure and key modules of proposed face recognition system is depicted in Figure . The face recognition has two parts serving different functions: learning and testing. In learning, the purpose is to build a classifier which can recognise human faces. The learning also includes feature extraction, feature selection and training classifier with a given face database. In face images from the face database, features are extracted by Gabor wavelet transform [6], and face images become Gabor wavelet representation. However, the dimension of representation is extremely high. It is impossible to manipulate directly. Hence, a feature selection module is applied to select a small set of Gabor wavelet features. With the selected features, face images in the database are projected into a smaller linear space, and feed to statistic classifiers (*e.g.* k-nearest neighbour) with their label. In testing, new faces are firstly gone through face and eyes detection and alignment operations. After projected with selected features, vectors representing new faces are sent to the trained classifier, and output the final decision. This paper will focus on feature selection in which multi-class AdaBoost learning is used to collect a small set of representative features, while feature extraction can be found in [7], [8].

A. *Multi-class AdaBoost*

AdaBoost is an efficient method for producing a highly accurate learning algorithm by combining a set of rough and moderately accurate learning algorithm. In term of AdaBoost, inaccurate learning algorithms sometimes are called "weak" learners, "weak" classifiers, or base classifiers. They have lower discrimination power to assign the true label correctly. Hence, AdaBoost refers to a method of combining a set of weak learners into a strong classifier which gives a high accuracy for prediction. AdaBoost has been a very prosperous approach for solving two-class classification. It also has two multi-class classification versions: AdaBoost.M1 (M1) [9] and AdaBoost.M2 (M2) [9]. M1 is the most direct way to perform multi-class classification, and M2 is an enhancement of M1.

Our multi-class AdaBoost algorithm is a combination of M1 and M2 which is a variant from M1 with some extension on multiple labels from M2. The algorithm of multi-class AdaBoost algorithm is given in Table I. In each iteration, a significant feature is selected with the lowest error ε_t, so that there are T significant features after T iterations. A multi-class weak learner $mh(x)$ is built on a single feature. The error ε is calculated by summing the weights of all misclassified examples. The importance α_t for each significant feature is evaluated by the lowest error ε_t in the t-th iteration.

B. *Multi-class Weak Learner*

In AdaBoost learning, weak learner is used to evaluate a feature. In [10], we have demonstrated that different weak

TABLE I
Multi-class AdaBoost algorithm

```
1: Given example $(x_1, y_1), \dots, (x_n, y_n)$, where $x_i$ is the data of
   the $i$th example, which are contributed by $k$ features $\{j_1, \dots, j_k\}$,
   and $y_i \in Y = \{1, \dots, c\}$ for $c$ subjects (classes)
2: Initialise the weights $\omega_{1,i} = \frac{1}{n}$ for each example $(x_i, y_i)$.
3: for $t = 1, \dots, T$ do
4:    Normalise the weights, $\omega_{t,i} \leftarrow \frac{\omega_{t,i}}{\sum_{i=1}^{n} \omega_{t,i}}$ so that $\omega_t$ form a
      probability distribution.
5:    for all $\{j_1, \dots, j_k\}$ do
6:       Train a multi-class weak learner $mh_j$ built with one single
         feature $j$ with the weights $\omega_{t,i}$.
7:       The error is calculated as $\varepsilon_j = \sum_{i=1}^{n} \omega_{t,i}\gamma$, where $\gamma = 1$
         when $y_i \notin mh_t(x_i)$, and $\gamma = 1$ otherwise.
8:    end for
9:    Choose the optimal multi-class weak learner $mh_t$ with the
      lowest error $\varepsilon_t$ from all $mh_j$.
10:   Select the corresponding feature $j_t$ of the multi-class weak
      learner $mh_t$ as a significant feature.
11:   Remove the feature $j_t$ from the feature set $\{j_1, \dots, j_k\}$.
12:   Update the weights $\omega_{t+1,i} = \omega_{t,i}\beta_t^{1-e_i}$, where $e_i = 0$ if
      $y_i \notin mh_t(x_i)$ and $e_i = 1$ otherwise, and $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$.
13: end for
```

learners lead to different performance of AdaBoost at the end. Hence, weak learners are crucial and primitive parts of the algorithm, and the design of weak learner is very important.

In this paper, a multi-class weak learner - **mPotsu** (multi-class Potsu) is proposed. The mPotsu weak learner is a multi-class variant from Potsu weak learner [11] which is a type of two-class classifier built with the concept of perceptron [12] and Otsu's thresholding algorithm [13]. The mPotsu is constructed by multiple Potsu weak learners with the *one-against-rest* strategy [14]. The strategy adopts a set of binary classifiers to create a multi-class classifier. Given an example x and its label in k classes, a binary Potsu weak learner is trained between a class j and test $k-1$ classes. Since there are k classes, an mPotsu contains k binary Potsu weak learners. Each Potsu gives an output indicating examples belonging to a class j or not.

By applying the multi-class algorithm in face recognition, the XM2VTS [15] face database is used to evaluate performance of the algorithm. In XM2VTS, since there are 200 clients (persons), *i.e.* 200 classes in the client set, an mPotsu applied in XM2VTS contains 200 binary Potsu learners. Each Potsu in an mPotsu is built for a particular class (client) against rest classes (clients). For example, the first Potsu in an mPotsu is built for the first client against all other 199 clients. Given an example x, the first Potsu tells x whether the first client or not. In XM2VTS, each mPotsu contains 200 binary Potsu learners.

The input vector of mPotsu is taken from a single Gabor wavelet feature across all examples in the training set. Hence, the input vector is one-dimensional. The training in each Potsu uses a heuristic approach to find an optimal threshold for separating the examples from the class j and other examples. If an example is successfully classified, the output will be labelled as 1, *i.e.* the positive. If not, the output will be labelled as 0, *i.e.* the negative.

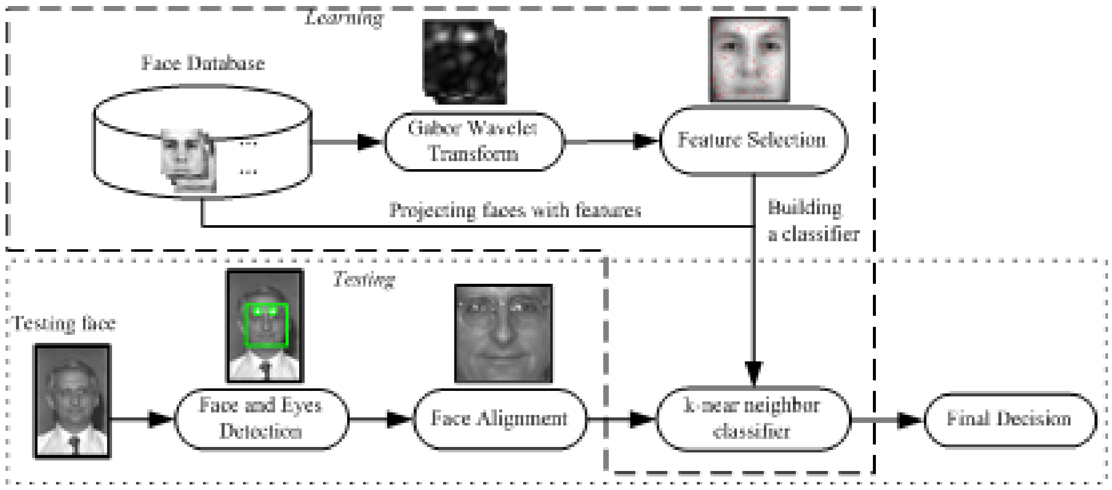

Fig. 2. The flowchart and key modules of the face recognition system

In XM2VTS, the decision making of mPotsu is to combine these 200 decisions. Ideally, if the training examples are well separated between each class, there will be only one positive output and 199 negative outputs. The plausible label is the corresponding class of positive label in mPotsu. However, in practice, there may be more than one Potsu weak learners giving positive labels. It is because the training examples are not linearly separable between different classes in most real situations. Hence, in the case of one-against-rest strategy, it is rare to output only one plausible label of multi-class classifiers. Instead of giving only one absolute label, mPotsu outputs a label vector η which compromises multiple positive labels.

$$\eta = (l_1, l_2, \ldots, l_k) \tag{1}$$

The label vector η contains k components corresponding to k classes. Each component l_j is corresponding to each class j, and has a boolean value between 1 and 0 indicating acceptance or rejection on the corresponding class j. By introducing the label vector η, mPotsu is able to coexist multiple decisions. For example, an mPotsu gives a label vector $\eta = (1, 1, 1, 1, 0, \ldots, 0)$, which indicates the plausible class might be class 1, class 2, class 3, or class 4 (since $l_1 = 1$, $l_2 = 1$, $l_3 = 1$, and $l_4 = 1$). It makes mPotsu not give an absolute decision, but several possible decisions.

In AdaBoost, after every weak learner is trained, an evaluation is needed to test the performance of the trained weak learner. Due to multi-label complexity on multi-class classification, a loose evaluation method is applied in mPotsu. Given an example with its true class j, if $l_j = 1$ in the giving label vector η, the classification will be considered as true positive no matter what other components $l_i = 1$. The classification is defined as

$$y \in mh_t(x) \quad \text{when} \quad l_y = 1 \tag{2}$$

where $mh_t(x)$ represents the mPotsu weak learner. For instance, an example x with its true label of class 1 is fed into an mPotsu. After training, the mPotsu gives a label vector $\eta = (1, 1, 1, 1, 0, \ldots, 0)$, where l_1 in the vector η is equal to 1, the classification is considered as true regardless $l_1 = 1$, $l_2 = 2$, $l_3 = 3$, and $l_4 = 4$. Since the method of classification is quite loose, the accuracy of mPotsu is low through assigning multiple labels to an example. However, since AdaBoost can boost the performance from a set of weak classifiers, the low accuracy on different mPotsu is accumulated into higher accuracy. After many iterations, the performance of multi-class AdaBoost is enhanced.

C. Computational Issue

Feature selection on multi-class AdaBoost is very time consuming, because each mPotsu contains 200 Potsu binary learners when operating in the XM2VTS database. The computational time on training an mPotsu is equivalent to time cost on training 200 binary Potsu learners individually. The AdaBoost algorithm searches over the whole feature set exhaustively in each iteration. Since the size of each face image is 27×28, and these images are convolved with 40 Gabor wavelet kernels in feature extraction to make magnitude responses, the total number of Gabor wavelet features is $27 \times 28 \times 40 = 30240$. In XM2VTS, the training dataset contains 800 face images, which are from 200 clients, *i.e.* 4 images per client. With a 2.8 GHz CPU, the training on a single mPotsu needs 11 seconds. One iteration in AdaBoost training takes $11 \times 30240 = 332640$ seconds which is roughly 93 hours. To select 200 features, it needs 200 iterations. The AdaBoost training takes $93 \times 200 = 18600$ hours, *i.e.* 775 days. The whole computational time is extremely long, and it makes feature selection hardly being accomplished with current computing facility. Hence, it is necessary to adopt Grid Computing technology to reduce the computation into affordable time.

IV. Implementation and Results

In this Section, the implementation of proposed algorithm is given. Firstly, a program is design to fit for grid computing. Secondly, it gives details on preparing executable program.

Finally, the results of features selected after Condor grid computing are presented.

A. From HTC to HPC

The aim of Condor grid is to provide researchers with a High Throughput Computing (HTC) resource. HTC is designed to process tasks that require fairly short processing times and independent to each other, but need to be run 100's or 1000's times. For example, a genetic algorithm process takes 30 minutes on a single node, but needs to be done 1000 times with different arguments. Therefore it requires 500 hours when running on only one node. If a Condor grid with around 250 nodes is used, the whole time can be reduced to 2 hours.

However, multi-class AdaBoost learning does not fall into the category of HTC. First of all, multi-class AdaBoost requires large amount of computational time. Secondly, it does not need run multiple times. Multi-class AdaBoost learning only needs a one-time running to find a small set of significant features for face recognition. It belongs to High Performance Computing (HPC) which is characterised as needing large amount of computing power for short periods of time. To employ Condor grid for the multi-class AdaBoost learning, it is necessary to alter the structure of learning algorithm for a HTC environment so that computation can be finished in a shorter time. Meanwhile, it will show that a HTC platform can be used for HPC with some alteration on source code.

The multi-class AdaBoost of feature selection is iteration-based learning. The learning is to run in 200 iterations for obtaining 200 significant features. The computation between iterations is sequential and dependent, since output of current iteration will be input of next iteration. Hence, it can not be solved distributable over the Condor Grid.

In each iteration, every feature is evaluated by the means of training an mPotsu with respect to the training accuracy. The evaluation on a feature does not require large amounts of time, and there are roughly 30000 features to be evaluated. Most importantly, evaluation between features is independent since output of evaluation on one feature is not influenced by other features. Hence, the conditions of HTC is satisfied by the feature evaluation within iteration. In Condor, one job could be an operation of evaluation on a specific feature, so that there would be over 30000 jobs generated and queued in the Condor pool. Too many jobs in the pool will bring excessive network transmission which makes transmission time is more than actual learning time. In some extreme cases, huge amount of jobs will lead to slow response on the master node, or result in deadlock over network. Therefore, the whole set of features is split into 120 subsets, and each subset contains 252 features. Every job is designed to evaluate 252 features from a specific subset, and there are 120 jobs in each iteration. Empirically, a job is normally done in around half an hour, since the nodes across the grid do not have the same CPU power. Some jobs finish rapidly due to the assigned node has more powerful CPU, while some jobs finish slowly due to less powerful CPU or jobs are interrupted and migrated to other nodes. If the Grid is only dedicated for AdaBoost learning at

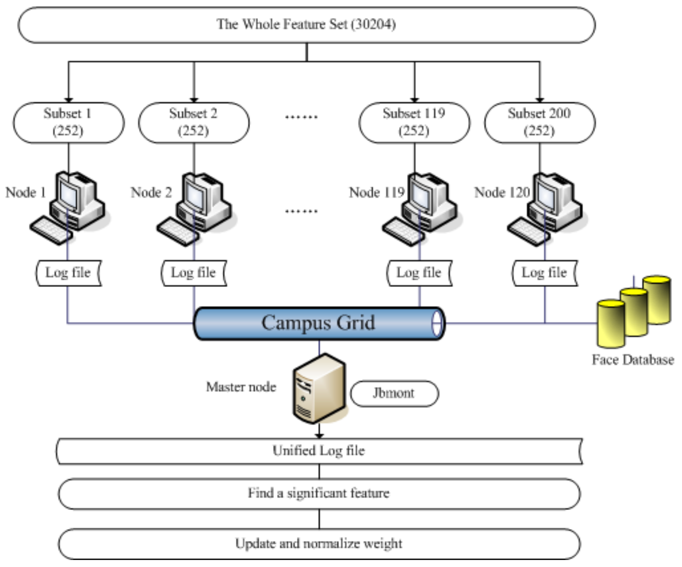

Fig. 3. The Grid computing diagram of Multi-class AdaBoost learning

the time of running, one iteration would also be finished in around half hours. In actually running, there could be running 120 nodes simultaneously for the best performance. Although more nodes are available, it is better to reserve some nodes for other users using grid computing.

Each job not only evaluate the given features, but also write the error of features into a log file in the shared NFS drive. On the master node, a program called *Jbmont* is monitoring progress of all running jobs in background. By scanning log files in a periodic time, Jbmont can get the running status of jobs. Once all 120 jobs for an iteration are finished, Jbmont immediately combines all log files into an unified file of all 30204 features. By automatic analysis upon errors in the unified file, Jbmont founds one significant feature and saves it in a file. After updating and normalising weights, Jbmont automatically generates a series of submission files, then submits these new jobs into the Condor pool for the next iteration. The diagram of Grid computing structure is displayed in Figure 3.

B. Program Preparation

An executable program designed for a stand-alone computer can not directly be deployed on the Condor Grid. The original source is need to be altered for running in the Condor grid. While running in the Condor grid, jobs may fail to complete due to exception of software, power cuts, system down, and other unpredictable reasons. Hence, jobs are designed to resume-able for avoiding restart from the beginning,. If a job is resumed, it is firstly to check the log file, and find where it fails and start with the place.

In this paper, the vanilla universe is chosen. During compiling source code, program is linked with static library. By doing so, program is more compatible with different Linux platforms. Since the program calls a set of external functions, routines and variables, *e.g.* OpenCV [16]. On some nodes, there may not have corresponding software installed. To prevent jobs from collapsing on nodes, the program is linked with static libraries. Normally, programs compiled with linking static-libraries normally takes more compiling time and more storage space.

C. Results

As mentioned before, an Intel Pentium 4 2.8 GHz CPU will take 775 days, *i.e.* roughly over two years. However, when the whole learning process is running on the Campus Grid, the computational time is reduced dramatically. It is assumed every node in the grid has the same computing capability as the Pentium 4 2.8 GHz CPU[1]. Running on each iteration will be finished in 46.2 minutes. To select 200 features, $9,240$ minutes (46.2×200), *i.e.* 154 hours are needed for 200 iterations learning. The whole computational time is less than one week.

Actually, the whole computation takes 20 days, since the grid is not mainly dedicated to scientific computing. Some nodes probably are not available after jobs are assigned to them. In this situation, jobs on them have to be manually removed, and are resubmitted to the Condor pool. Meanwhile, not all nodes have the same hardware configuration which leads some jobs finished earlier, but others finished later. In addition, there might be some miss links or temporary disconnection which lead to failure in accessing images from face database. Hence it is very difficult to predict accurate computational time for grid computing. However, comparing to over two years computation, the grid computing demonstrates its significant advantages over the conventional computing.

After 200 iterations, 200 features are shown in Figure 4. These 200 features are general for all 200 clients in the XM2VTS face database. The feature projection plane in Figure 4 is a mean face image generated from the training set.

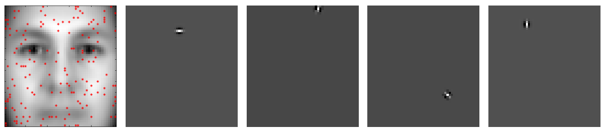
Fig. 4. The 200 features on face and the first four features

V. Conclusion

In this paper, we have presented a Condor grid computing solution for multi-class AdaBoost learning of feature selection. Condor is a HTC software specialised in job and resource management. The multi-class AdaBoost of feature selection is an iteration-based learning on exhaustive evaluation on a whole feature set. Each feature is evaluated by training performance of mPotsu weak learner which is constructed by multiple binary Potsu weak learners. To implement the multi-class learning in grid computing, we have improved the learning algorithm so that Condor is successfully used for the HPC purpose. Selecting 200 features on the Condor grid computing is finished in 20 days, while running the feature selection on a single CPU requires over two years. By using Condor grid computing, the total computational time is reduced into approximately $1/n$ of the original time (n is the number of nodes) in theory. It is indicated that grid computing can significantly reduce the computational time of machine vision applications. The only problem of Grid Computing is hard to assess the performance. Since not all nodes are only dedicated for Grid Computing, it is very difficult to predict accurate computational time. Nevertheless, Grid Computing is an excellent tool for HPC by small organisations or research sectors which can not afford supercomputing facilities.

[1]Actually, on Reading campus grid, there are many nodes with Intel Core 2 CPUs which are faster than Intel Pentium 4 CPUs.

References

[1] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 2001, pp. 511–518.

[2] D. Young and J. Ferryman, "Faster learning via optimised AdaBoost," in *IEEE Conference on Advanced Video and Signal Based Surveillance*, 2005.

[3] R. Buyya and S. Venugopal, "A gentle introduction to grid computing and technologies," *CSI Communications*, vol. 29, no. 1, pp. 9–19, 2005.

[4] D. Thain, T. Tannenbaum, and M. Livny, "How to measure a large open source distributed system," *Concurrency and Computation: Practice and Experience*, vol. 8, no. 15, 2006.

[5] D. Aloni, "Cooperative linux," in *Proceedings of the Linux Symposium*, 2004.

[6] M. Zhou and H. Wei, "Face verification using gabor wavelets and AdaBoost," in *Proceedings of 18th International Conference on Pattern Recognition*, 2006.

[7] ——, "Face feature extraction and selection by gabor wavelets and boosting," in *Proceedings of International Congress on Image and Signal Processing*, 2009.

[8] M. Zhou, *Gabor-Boosting Face Recognition: From Machine Learning Perspective*. VDM Verlag Publishing, 2009.

[9] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *Proceedings of the Second European Conference on Computational Learning Theory*, 1995.

[10] M. Zhou and H. Wei, "Constructing weak learner and performance evaluation in AdaBoost," in *Proceedings of International Conference on Computational Intelligence and Software Engineering*, Dec. 2009.

[11] ——, "Heuristic weak learner in AdaBoost for face recognition," University of Reading, Tech. Rep., 2009.

[12] S. I. Gallant, "Perceptron-based learning algorithms," *IEEE Transactions on Neural Networks*, vol. 1, no. 2, pp. 179–191, 1990.

[13] N. Otsu, "A threshold selection method from gray level histograms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 9, pp. 62–66, March 1979.

[14] D. Tax and R. Duin, "Using two-class classifiers for multiclass classification," in *International Conference on Pattern Recognition*, vol. 16, 2002, pp. 124–127.

[15] K. Messer, J. Matas, J. Kittler, and K. Jonsson, "XM2VTSDB: The extended M2VTS database," in *Audio- and Video-based Biometric Person Authentication, AVBPA'99*, Washington, D.C., March 1999, pp. 72–77, 16 IDIAP–RR 99-02.

[16] "OpenCV." [Online]. Available: http://opencv.willowgarage.com/