Resource Use Pattern Analysis for Opportunistic Grids*

Marcelo Finger†
Dept. of Computer Science
University of São Paulo, Brazil
mfinger@ime.usp.br

Germano C. Bezerra
Dept. of Computer Science
University of São Paulo, Brazil
germanob@ime.usp.br

Danilo R. Conde
Dept. of Computer Science
University of São Paulo, Brazil
danconde@ime.usp.br

ABSTRACT

This work presents a method for predicting resource availability in opportunistic grids by means of Use Pattern Analysis (UPA), a technique based on non-supervised learning methods. The basic assumptions of the method and its capability to predict resource availability were demonstrated by simulations; accurate learning techniques and distance metrics are determined. The UPA method was implemented and experiments showed the feasibility of its use in low-overhead scheduling of grid tasks and its superiority over other predictive and non-predictive methods.

Categories and Subject Descriptors

C.2 [**Computer-communication Networks**]: Distributed Systems; I.5 [**Pattern Recognition**]: Clustering

General Terms

Algorithms, Performance

Keywords

Use Pattern Analysis, Scheduling, Opportunistic Grids, Grid Computing

1. INTRODUCTION

Grid computing deals with computationally intensive distributed applications on heterogeneous environments. Opportunistic grids differ from dedicated grids in that the machines may not be always available to run grid tasks. In *opportunistic Grid Computing*, grid applications use the idle time of desktop machines to perform high-performance computation. If a grid application is running when the machine is claimed back by its owner, the grid job is either interrupted, migrated or simply aborted. Resource owners have to allow the use of the idle time of their computational resources by the grid; but they will only do it provided that the perceived compromise on the Quality of Service for their private computations remains very low.

Research groups working on grid systems, such as Condor [10], BOINC [1], OurGrid [3], and our own work on InteGrade [6] have investigated opportunistic grid computing to perform high-performance computation. However, the support for effectively using these shared resources without compromising the Quality of Service perceived by the resource owners is still very limited.

To address the problem of effective opportunistic computation, it is very useful to be able to predict when a given computational resource will be idle, becoming available for grid applications. If this kind of prediction can be done with some degree of accuracy, the scheduler may be more effective in its task of assigning jobs to machines. Each resource has its own use pattern, and the *Use Pattern Analysis* (UPA) consists of the task of detecting the local use pattern of each resource in each machine.

In this work, a method is described aiming at performing resource use pattern analysis for machines belonging to opportunistic grids. This method is based on unsupervised machine learning [12, 2], which performs a *clustering analysis* on past records of resource use to discover prototypical patterns of use [7]. As usual in machine learning settings, the learning activity is performed off-line. The use patterns learned are used at run-time to predict the availability of computational resources.

The development of the method was done in two steps: simulation and implementation. A simulation phase was needed due to the existence of a large number of parameters to be set in clustering-based unsupervised machine learning methods, and was performed using real data collected from machines belonging to the InteGrade opportunistic grid [6]. Our group developed the *Local Use Pattern Analyser* (LUPA) module for the InteGrade grid, and validated the results of the simulation. The implementation was used to compare the proposed method against other methods for availability prediction.

The rest of the paper develops as follows. The UPA method is described in Section 2. Its basic assumption is validated in the simulation of Section 3 which also determines several parameters that improve the accuracy of the method. Implementation experiments are described in Section 4. Conclusions and further work are presented in 5.

1.1 Related work

Research work on prediction about grid tasks has con-

*Supported by CNPq/Brazil project 550895/2007-8.

†Partly supported by CNPq, grant PQ 304607/2007-0.

MGC'08, December 1–5, 2008 Leuven, Belgium

centrate on scheduling methods to improve grid application execution time [11, 13].

Yang, Schopf and Foster [15] propose a *conservative scheduling* technique that uses predicted mean and variance CPU capacity information to make data-mapping decisions. It employs one-step-ahead CPU load prediction based on history CPU load information.

The Condor Grid system [14, 10] does not predict, but detects when an application performance is very slow in a machine and migrates it to some other machine on the grid.

The BOINC system [1] has very recently taken a direction that most aligns with ours. It uses clustering as a means to identify correlated availability of Internet resources [8], based on independent but similar ideas than the ones developed here. That work differs form ours in that it tries to identify patterns of availability searching for correlations between machines over the whole Internet, which implies in a considerable data collection and processing task, while our work identifies local patterns of availability, internally to each machine, at very low cost.

2. THE UPA METHOD

The resource Use Pattern Analysis (UPA) method is based on the assumption that there is a small set of prototypical daily behaviours that models resource availability at each machine. As this is a somewhat bold assumption, prior to implementation a simulation phase was needed to both validate and fine-tune the method.

Use Pattern Analysis deals with machine resource use objects. Each object is a vector of values representing the time series of a machine resource use, as illustrated in Figure 1. Machine resource use is sampled at a fixed rate (currently, once every 5 minutes) and grouped in objects covering 48 hours. An object starts at midnight, so there is a 24-hour overlap between consecutive objects. The span of 48 hours, instead of 24 hours, is needed for the prediction phase.

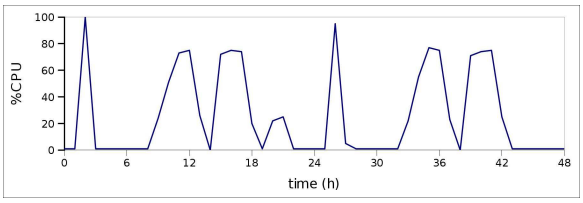

Figure 1: An object representing a machine's CPU use over a 48h period

The method is capable of monitoring several computational resources, such as CPU use, available RAM, disk space, swap space, network and disk I/O; in this work, only the first two will be discussed, and these are the most relevant for machine allocation decisions. Use Pattern Analysis performs unsupervised machine learning [12, 2] via a data clustering process [7] to obtain a fixed number of *use classes*, where each class is represented by its *prototypical* object. The idea is that each class represents a frequent use pattern, such as a busy work day, a light work day or a holiday.

The method involves two phases: a training/learning phase and an execution/prediction phase.

2.1 Learning

Learning is an off-line activity that inputs a large amount of objects collected during the machine regular operation. A clustering algorithm is applied to the training data [7, 5], which groups the training objects into a fixed number k of clusters. At each cluster, a prototypical element is computed that represents the whole use class. The output of the method are k prototypical vectors to be used by the runtime predictor.

It is often common to have incomplete data for training, due mainly to machine down-time. One possible way to deal with this problem in the literature is to apply an Expectation Maximisation completion process [4]. However, the current approach opted for a simpler solution, and discarded objects with incomplete data. Provided there is sufficient data, this solution does not affect the result. As a consequence, resource use is equivalent to its unavailability.

In fact, this form of learning is only a reliable method when there is a considerable mass of data, which in this case consisted of at least 60 objects, or two months of data. At the implementation phase, other methods were tested that need smaller amount of testing data, and their results were compared against the UPA method.

Data clustering analysis can be parameterised in several ways. For the UPA development, the following parameters were considered:

- *Number of clusters.* We considered a fixed number of clusters, either 5 or 10.
- *Data normalisation.* Two possible data normalisation schemas were considered for vector: *no normalisation*; and *variational* normalisation, in which the means of all points in a vector is subtracted from all data.
- *Computation of prototypical element.* Typically this is a *centroid*, that is, the average of all cluster elements, or the *centre*, that is, an element of the cluster closer to the centre. Only the centroid method was considered.
- *Similarity measurement.* There are several ways to compute the distance between two clusters (similarity is the inverse of distance), assuming that the distance between any pair of points a and b is given by the Euclidean distance: $d_{ab} = \sqrt{\sum(a_i - b_i)^2}$. The following distances were considered: *single linkage*, the smallest distance between points; *complete linkage*, the largest distance; *centroid method*, the distance between cluster centroids. *Ward's method*, the variance of the union of the clusters.
- *Clustering algorithms.* There are many algorithms for clustering, namely hierarchical, sequential, k-means, etc. No substantial difference in prediction power was noted among different algorithms. To concentrate on distance measurements, only hierarchical algorithms are considered, which constructs clusters on a bottom-up fashion, each step uniting the two closer (ie, more similar, less distant) clusters.

2.2 Runtime Prediction

During runtime, a request is sent to each machine predictor specifying the amount of resources (CPU, disk space, RAM, etc) and the expected duration needed by an application to be executed at that machine; this expected duration may be provided by users, which may not be a reliable

estimate [9]; work in progress investigates the automated learning of program resource needs. The UPA predictor has to decide if this machine will be available for the expected duration.

This decision is reached according to the following method. A record is kept about the recent use of each resource; usually the last 24 hours, as illustrated in Figure 2. The predictor computes the *distance* between the vector representing recent history and each of the prototypical element of the use classes learned during the training phase. The recent use object has span of 24h, but the prototypical elements have span of 48h. To compute the distance, the recent record of resource use is compared with the corresponding times in the use classes; so if a request for a resource is made at 6 pm, the last 24 hours of that resource use is compared to the interval 18–42 in each prototypical element. The class with the smallest distance is the *current use class*.

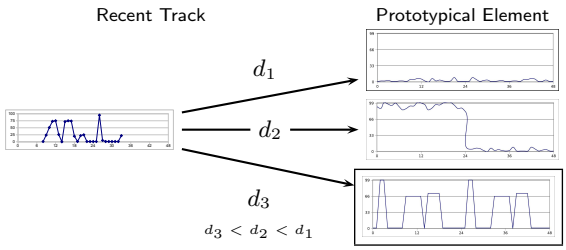

Figure 2: Prediction of class pertinence

The interval between the current time and the end of the 48h in the current use class, after some possible processing, is used as the prediction of the near future; so if a request is made at 6pm, this method can predict the next 6 hours. When variational normalisation is used, the average of the recent track has to be added to obtain a prediction. If all requested resources are predicted to be available, then the application is scheduled to be executed; otherwise, it is rejected.

If a prediction for a longer period is needed, there are two possibilities:

(a) To use less that 24 hours of recent record to compare with the start of each prototypical element, which allows for predictions over 24h. The reliability of predictions decreases with the span of the recent record used.

(b) To chain predictions, by using the last 24 hours of record or prediction as a basis for the next prediction. This allows for unbounded perditions, but the longer the chain, the less reliable the prediction. This is not implemented yet and remains for future work.

3. SIMULATION

There are many parameters involved in data clustering analysis. Simulation was used to choose those parameters. In the end, with a set of chosen parameters, the basic assumption of the UPA method could be evaluated, namely that prototypical daily behaviours are good models of resource availability and are worthy of being implemented.

The data for simulation was collected, for CPU and RAM use, during a period of 120 days from four Linux machines with very different types of users:

(a) a general purpose machine with more than 30 users;

(b) a single user machine;

(c) a general purpose machine with 6 users;

(d) a multi-user machine employed for testing heavy computational linguistics programs.

For each machine, the learning process generated both a 5-cluster and a 10-cluster output, called according to the presentation above a5, a10, b5, b10, c5, c10, d5, d10; furthermore, clustering was performed both for pure and for normalised data.

A single test is defined by a resource r, an instant in time t, an availability level α and an interval ι; the test output is *yes* if it is predicted that the availability level of r will remain above α for duration ι starting at t, and *no* otherwise. The output is *correct* if the decision coincides with that obtained from simulation data.

An *availability matrix* $A = \{a_{ij}\}_{N \times M}$ was constructed for each machine and each resource, with availability α in the interval 10%–90% of total capacity with 10% steps, and for intervals ι of $10, 20, 30, 60, 90, 120, 240$ minutes. Each a_{ij} contains the percentage of correct predictions in 100 tests for a given pair $\langle \alpha, \iota \rangle$, with random starting point t. For each matrix A, the average prediction success was computed as $\mu_A = \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} a_{ij}$.

One simulation experiment consists of the construction of an availability matrix A containing 63 cells, each cell the result of 100 tests, generating a single value for the average prediction success, μ_A. The simulation experiments were made in batches of 8 experiments, one for each machine with 5 and 10 clusters. For each resource, 4 batches of experiments were made, each using a different distance measurement. The 64 experiments were repeated for normalised and non-normalised clusters. Overall, 128 experiments are reported.

3.1 Simulation Results

Figure 3 shows the results of all experiments, separating the results of predictions based on pure and normalised (variational) data. The values of μ_A were surprisingly high. All values for average prediction success were above 75%, and mostly above 80%.

The quality of the results allows for the validation of the basic assumption of the UPA method.

Besides validating the UPA assumption, Figure 3 compares the prediction results obtained from clustering process with pure data and from normalised clustering with 0-mean, variational data. Predictions with normalised data yield a better result in all experiments, with average prediction success were above 90% in all cases.

For this reason, the following results are presented only for normalised clustering.

Experiments were made for hierarchical clustering using four different types of distance measurements, as described in Section 2.1. Figure 4 presents the results for RAM availability prediction and Figure 5 presents the results for CPU availability prediction.

No distance measurement dominates all the others all the time. However, it seems that computing distances by Ward's method has a slightly superior overall performance. Similarly, no measurement is clearly the worst. As the number of machines analysed is still small, no categorical preference can be established. The results in Figures 4 and 5 may even

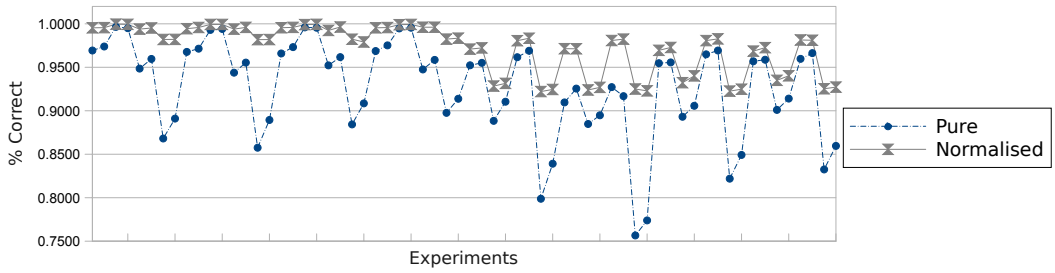

Figure 3: Success rates of all experiments, pure × normalised data

suggest that each domain of machines may require an initial analysis to determine the best measurement for each case.

It is worth noting that predicting RAM availability is easier than predicting CPU use. It calls the attention that machine (b) yields the best results for RAM prediction and the worst ones for CPU prediction. This reinforces the need for an initial *per machine* analysis prior to the choice of measurement to be implemented.

Figures 4 and 5 show a consistent prevalence of 10-cluster predictions over 5-cluster prediction for all distance measurements. However, in all cases, this prevalence is in fact very small. Furthermore, by investigating the clusters learned with 10-cluster outputs, one verifies that a considerable number of clusters has a very small number of elements, normally less than 5. On the other hand, with 5-cluster outputs the representativity of the clusters learned is always higher.

So no categorical conclusion can be reached as to the fact on the number of clusters, other than that there no big difference is achieved by choosing a larger number of clusters.

3.2 Simulation Conclusions

Simulation definitely validated the UPA assumption. Data normalisation is to be preferred, and the number of clusters can be kept as low as 5, without seriously affecting prediction performance. Distance measurements have to be analysed on a case-by-case prior to implementation, but no method among the analysed ones is either the best or the worst.

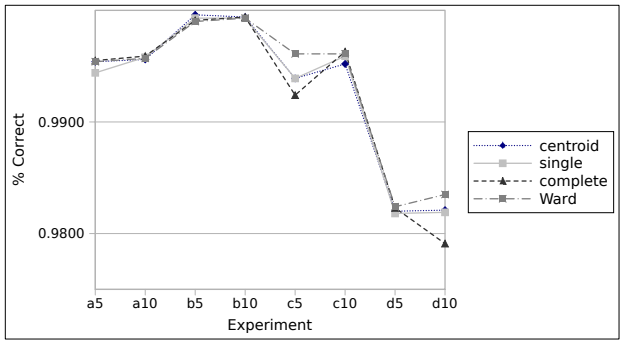

Figure 4: RAM Availability Prediction

4. IMPLEMENTATION

The development of an implementation of a resource Use Pattern Analyser had three main goals:

- to explore ways the scheduling of grid applications using the UPA method;

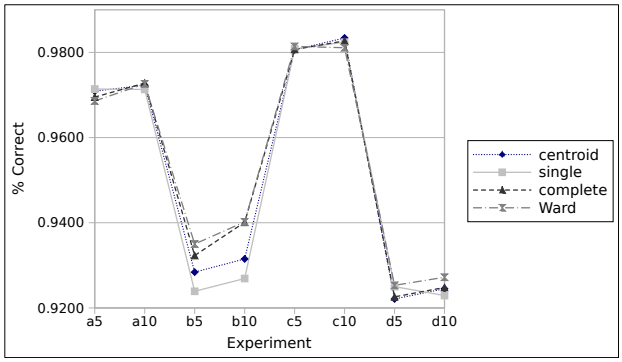

Figure 5: CPU Availability Prediction

- to compare the UPA scheduling with other scheduling methods.
- As the UPA method needs a few months of resource use data collection to be reliable, identify initialisation strategies to be used during the first period of activation.

4.1 Design Decisions

Resource use data information has to be collected locally at each machine taking part in an opportunistic grid. An initial design decision concerns whether this data is to be analysed locally, so that the resource use patterns is hidden from the rest of the system, or if such data is to be sent to an external analyser. The latter possibility may make the task of machine allocation easier, but it can be seen as a breach on the privacy by users of single-user machines, which may not allow those machines to take part on the grid for fear that their pattern of work is being monitored. The local analysis of data also has the advantage of preventing the flow of use data on the network, decreasing the overhead placed on the system.

So a Local resource Use Pattern Analyser (LUPA) module is installed in all machines that allow their resource to be opportunistically used by grid applications. The LUPA module was implemented in C++ on several Linux platforms, and is now an integral part of the InteGrade distribution [1].

The LUPA architecture is shown if Figure 6, and consists of three submodules:

- **Data Collection**. Performs resource use sampling, recording CPU and RAM use every 5 minutes. This is

[1]Freely available at `http://www.integrade.org.br`.

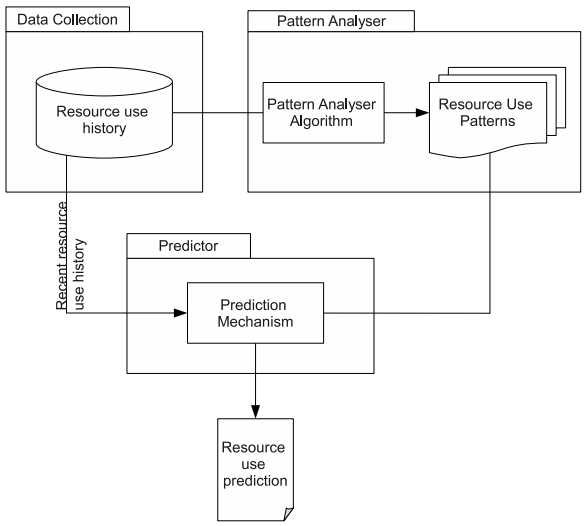

Figure 6: The LUPA Architecture

the same rate used for simulations.

- **Pattern Analyser**. Performs off-line clustering of resource use objects, as described in Section 2, generating a fixed number k of prototypical use classes; for the results presented below, $k = 5$. For time-efficiency reasons, centroid distance is used.

  Currently, patterns are recomputed once a day but that, as will be shown, is not a burden on the system.

- **Predictor**. Performs runtime predictions based on the recent resource use history. The simple interface to access the predictor is:

  `double[] getPrediction(resource r, int hours);`

  which returns a vector of values representing the `r`-use prediction for the next `hours`, in 5-minute intervals.

It is important to note that at this stage of development, the LUPA module is not yet integrated with the grid scheduler. In the experiments for evaluating the implementation, scheduling is restricted to selecting the machine with higher CPU availability.

4.2 Experiments

The experiments simulate the scheduling of a grid application. Experiments were performed using 15 machines, with data collection logs varying from 41 to 120 days, none of which were used for the simulation phase, so as to avoid biases results.

Three different scheduling methods were compared:

- **RR: round robin.** Machines are randomly placed on a circular list; when n machines are requested, the initial n machines are returned and the list starting pointer is advanced n positions. No prediction is made.

- **last4.** Resource prediction for any amount of time is considered the average of the last four hours. The method chooses n machines with lower prediction of resource use.

  Preliminary tests were also made for the last 8, 12 and 24 hours, and their performance were considered basically equivalent to the **last4** method.

- **UPA.** Prediction using the UPA method. The method chooses n machines with smaller prediction of CPU use for the next h hours.

An experiment consists of a sequence of tests for a fixed set of tests parameters. Initially, the valid days in the data collection logs are selected, in which there are at least m valid machines, where m is a test parameter. For each such day, pattern analysis is run and then 24 tests are executed, each for an hour of the day and for each to the three scheduling algorithms above. These are the *instant tests*.

The performance metric is the average free CPU in the chosen machines in the interval $[t, t + h]$, given by

$$performance_s(t) = 1 - \frac{\sum_{i=1}^{n} use(m_i, t, h)}{n}$$

where s is the scheduling algorithm, n is the number of chosen machines, m_i is one of the machines chosen to run an application process for h hours starting at time t, and $use(m_i, t, h)$ is the average CPU use at machine m_i for that period according to the log. Experiments can constrain the set of chosen machines by requiring a minimum CPU availability a at chosen machines; if this requirement is not met, the instant test is not computed.

An experiment set of parameters is therefore given by:

- n: number of machines to be chosen to run the application, $1 \leq n \leq 6$;
- h: application duration, in hours, $h \in \{2, 4, 6, 12, 24\}$;
- m: minimum number of machines available to be chosen at a given day, $m \geq n$ and $1 \leq m \leq 15$;
- a: minimum CPU availability at chosen machines, $a \in \{0.6, 0.7, 0.8, 0.9, 0.98\}$.

An experiment is performed for a set of parameters only if the logs allowed for at least 100 instant tests for that set. A total of 45 experiments were made, averaging 527 tests each. The output of an experiment is the performance of each of the three scheduling methods for each test. The performance of scheduling methods s and r were compared on a test by test basis. Methods s_1 and s_2 were considered *equivalent* for a test if the performance difference was less than 2%, that is, $|performance_{s_1}(t) - performance_{s_2}(t)| < 0.02$. Method s_1 performs better than method s_2 if $performance_{s_1}(t) - performance_{s_2}(t) > 0.02$.

Results. The performance of **RR** was consistently below that of both **last4** and **UPA** scheduling methods, beating **last4** on less than 1% of the tests and **UPA** even less.

Figure 7 displays a comparison between the **UPA** and **last4** scheduling methods showing, for each experiment, the percentage of tests one method is better than or equivalent to the other. It calls the attention that for an average of 77.6% of all tests, the two methods are equivalent. When equivalent results are discarded, the **UPA** performs better than **last4** in 75.6% of the experiments.

Table 1 summarises the effect of the test parameters on the performance of the methods. Scheduling using **UPA** has its relative performance increased when the ratio m/n

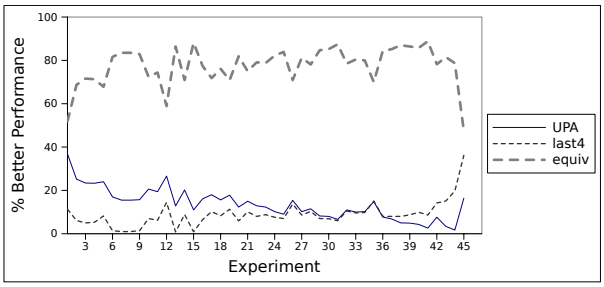

Figure 7: UPA *versus* last4

	UPA	**last4**
Performance	Similar to **last4**, but better	Similar to **UPA**, but worse
Impact of $\uparrow m/n$	$\uparrow$ performance	no effect
Impact of $\uparrow d$	$\uparrow$ performance w.r.t **last4**	$\downarrow$ performance w.r.t. **UPA**
Impact of $\uparrow h$	no effect	no effect

Table 1: Scheduling Methods Comparison Summary

(available/required machines) increases, as well as when d (required free CPU) increases. Overall, the **UPA** method displays better results. It was also clear that the method performs well with less than 60-day training data.

We conclude that **last4** is quite a good method, and a candidate to be run during LUPA initialisation. Final LUPA implementation has the following adaptative behaviour:

- Use **UPA** method if more than 21 days of data collection is available; else
- Use **last4** method if more than 4 hours of data collection is available; else
- Predict that resource use at request time persists.

With regards to system overhead, the running time for pattern analysis was always below 1s, and the running time for prediction calls was always below 3ms. Measurements were made on a notebook with an AMD Turion 64 1.8GHz CPU, 1GB RAM running Kubuntu 7.10 (32 bits) Linux.

5. CONCLUSIONS AND FURTHER WORK

The experiments have shown that some form of prediction always perform better than no prediction, and the UPA-scheduling method was favourably compared with respect to other methods, with small overhead. This confirms that the method can be used in practical scheduling of grid application tasks.

Future work includes integrating the LUPA module with a task scheduler, so that experiments can be made using real grid applications and several scheduling variants applying Use Pattern Analysis can be tested. Furthermore, it remains to be explored the capacity for longer term predictions, its reliability; there are several scheduling possibilities this capacity allows for, which deserves a detailed analysis, such as preemptive task migration and the automated "booking" of machine resources for future executions. Also, the existence of long histories for training has to be studied, to determine the ideal weight to be given to recent and distant past information.

6. REFERENCES

[1] D. P. Anderson. BOINC: A system for public-resource computing and storage. In *GRID '04: 5th IEEE/ACM Int. Workshop on Grid Computing*, pages 4–10, 2004.

[2] H. B. Barlow. Unsupervised learning. In G. Hinton and T. J. Sejnowski, editors, *Unsupervised Learning: Foundations of Neural Computation*, pages 1–17. MIT Press, 1999.

[3] W. Cirne, F. Brasileiro, N. Andrade, L. Costa, A. Andrade, R. Novaes, and M. Mowbray. Labs of the world, unite!!! *Journal of Grid Computing*, 4(3):225–246, September 2006.

[4] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[5] B. Everitt, S. Landau, and M. Leese. *Cluster Analysis*. Hodder Arnold, 4th edition edition, 2001.

[6] A. Goldchleger, F. Kon, A. Goldman, M. Finger, and G. C. Bezerra. InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines. *Concurrency and Computation: Practice and Experience*, 16(5):449–459, March 2004.

[7] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, September 1999.

[8] D. Kondo, A. Andrzejak, and D. P. Anderson. On correlated availability in internet-distributed systems. In *9th IEEE/ACM International Conference on Grid Computing (Grid 2008)*, 2008.

[9] C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely. Are user runtime estimates inherently inaccurate? In *JSSPP*, volume 3277 of *Lecture Notes in Computer Science*, pages 253–263, 2004.

[10] M. Litzkow, M. Livny, and M. Mutka. Condor - A hunter of idle workstations. In *ICDCS '88: Proceedings of the 8th Int. Conference of Distributed Computing Systems*, pages 104–111, June 1988.

[11] N. T. Loc, S. Elnaffar, T. Katayama, and H. T. Bao. Grid scheduling using 2-phase prediction (2pp) of cpu power. *Innovations in Information Technology*, 150, issue 2:1–5, 2006.

[12] T. M. Mitchell. *Machine Learning*. Computer Science Series. McGraw-Hill, 1997.

[13] M. Silberstein, D. Geiger, and A. Schuster. Scheduling mixed workloads in multi-grids: the grid execution hierarchy. In *Proceedings of the 15th International Symposium on High Performance Distributed Computing (HPDC06)*, pages 291–302, 2006.

[14] D. Wright. Cheap cycles from the desktop to the dedicated cluster: Combining opportunistic and dedicated scheduling with Condor. In *Proceedings of Linux Clusters: The HPC Revolution*, 2001.

[15] L. Yang, J. M. Schopf, and I. T. Foster. Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments. In *Proceedings of the ACM/IEEE Super Computing Conference*, page 31, 2003.