Computing and Informatics, Vol. 33, 2014, 1001–1027, V 2014-Oct-21

PFS: A PRODUCTIVITY FORECASTING SYSTEM FOR DESKTOP COMPUTERS TO IMPROVE GRID APPLICATIONS PERFORMANCE IN ENTERPRISE DESKTOP GRID

Sergio Ariel SALINAS

ITIC Research Institute
Universidad Nacional de Cuyo
ECT, Padre Jorge Contreras 1300
M5500 Mendoza, Argentina
e-mail: `ssalinas@itu.uncu.edu.ar`

Carlos GARCÍA GARINO

ITIC Research Institute & Facultad de Ingeniería
Universidad Nacional de Cuyo
Centro Universitario
M5500 Mendoza, Argentina
e-mail: `cgarcia@itu.uncu.edu.ar`

Alejandro ZUNINO

CONICET – ISISTAN Research Institute
UNICEN University
Paraje Arroyo Seco
B7001BBO Tandil, Buenos Aires, Argentina
e-mail: `azunino@exa.unicen.edu.ar`

Abstract. An Enterprise Desktop Grid (EDG) is a low cost platform that gathers desktop computers spread over different institutions. This platform uses desktop computers idle time to run grid applications. We argue that computers in these environments have a predictable productivity that affects a grid application execution time. In this paper, we propose a system called PFS for computer productivity forecasting that improves grid applications performance. We simulated 157 500 applications and compared the performance achieved by our proposal against two recent strategies. Our experiments show that a grid scheduler based on PFS runs applications faster than schedulers based on other selection strategies.

Keywords: Enterprise Desktop Grid, computer productivity forecasting, prediction methods, grid scheduling systems, heuristic method

Mathematics Subject Classification 2010: 68M20, 68M14, 68T01

1 INTRODUCTION

In recent years, an Enterprise Desktop Grid (EDG) [30, 25] has become a less expensive alternative to traditional computational grids. This technology is a type of volunteer computing [28] that gathers desktop computers spread over different institutions all over the world such as universities, schools and enterprises. Currently, volunteer computing systems provide several PetaFLOPS of computational power to solve scientific problems as shown in many publications supported by the BOINC [27] system. An EDG takes advantage of idle time from desktop computers voluntarily donated by users to run computation-intensive applications with independent tasks a.k.a. grid applications. Since computers are not fully dedicated to grid computing purposes a grid application can be suspended anytime by users. For this reason, it is important to make an optimal usage of the computational power provided by idle computers to improve grid applications runtime.

In volunteer systems, a grid application runtime not only depends on a computer capability but also its idle time. A grid scheduler [23, 39, 40] may improve its performance if it takes this into account. For instance, a grid scheduler queries a Resource Discovery System (RDS) [18] for a set of computers according to certain computational capabilities constrains. Based on the RDS response, it starts a computer selection process. At this point, we assume that candidate computers have the same computational capabilities. However, these computers will finish running a grid application at different times because they have different idle times to execute grid applications. Details about how a grid scheduler and Resource Discovery System works are out of the scope of this work.

In EDGs, computers are not fully dedicated to the computational grid; for this reason, any computer will run a grid application whenever that computer is idle. Many works deal with the problem of predicting whether a computer will be available or not to start running an application. Nevertheless, we did not find in the literature a work that addresses the following question: "*Is it possible to improve a grid application runtime in a time-shared environment such as volunteer systems?*". In this sense, a computer has different and changing statuses (idle, busy or off) that affects grid application runtime.

We use the term "*computer productivity*" previously defined in [41] to denote the efficiency with which a computer processes a grid application. We will show in Section 3 how that efficiency is affected by computer status changes. If a computer remains idle for a long period of time it will be able to execute an application faster than a computer that constantly switches its status from idle to busy. In this situation, a grid scheduler may improve application runtime by selecting computers based on their productivity. The research problem is "*how to calculate and forecast a computer productivity to improve the computer selection process made by a grid scheduler*".

In this work, we propose to address this problem based on the following strategies:

1. to calculate a computer productivity that represents the relationship between computer status changes and a grid application performance,
2. to forecast a computer productivity to estimate how it would affect a grid application performance and
3. to provide this information to a grid scheduler to choose computers that are expected to execute applications faster.

To calculate how a computer productivity affects an application performance is not an easy task. We assume that computer status changes occur at discrete instants of time. We estimate the relationship between status changes and a grid application performance by analyzing computer activity traces. As result, we have designed a heuristic method that calculates a computer productivity based on a set of metrics. Finally, we propose to forecast a computer productivity having as input an estimation of future computer statuses.

In EDGs, desktop computer status changes show certain well-defined patterns [13]. Computers located at the same office stay on at the same periods of time, for instance from 08:00 to 18:00. On the other hand, computer usage at laboratories is organized according to a teaching schedule. This predictable usage pattern facilitated the development of accurate prediction methods [2, 34, 26, 15]. Nowadays, it is feasible to predict a sequence of expected computer statuses for a long discrete period of time [14, 9]. Consequently, it is possible to forecast a computer productivity based on this information.

There are grid schedulers that use metrics in their selection process to improve its performance [16, 31]. In contrast to these works, we propose a system to forecast a computer productivity in EDGs based on a heuristic method. In a previous work, the authors introduced a preliminary version of our proposal [37], from now on called PFS (Productivity Forecasting System). Encouraged by the preliminary empirical validation, this paper presents an improved system and further experiments. We did not find a similar approach in the literature. However, our proposal aims to improve a grid scheduler performance by selecting computers based on a forecasting of their productivity. For this reason, we compare PFS against two current computer selection strategies:

1. an existing work [16] where computers are selected based on a metric called resource availability prediction (R.A.P.) and
2. random computer selection.

To validate our proposal, we ran experiments based on monitored information from a set of desktop computers at UNCuyo University[1]. We developed a grid scheduler simulator that uses three different computer selection strategies:

1. computer selection based on PFS,
2. computer selection based on RAP and
3. random computer selection.

We run 157 500 application instances and evaluated the performance achieved by the three strategies. The results showed that grid applications scheduled using PFS perform up to 80 % faster than other approaches.

Bear in mind that our proposal was validated in a simulation environment. For this reason, the integration of the PFS with any current EDG implementation requires an analysis that is part of future works.

The main contributions of this work are the following:

1. to validate that grid applications can run faster on EDGs when a grid scheduler selects computers based on their expected productivity and
2. we empirically show that it is possible to calculate and to forecast a computer productivity based on a sequence of that computer statuses estimated with any prediction technique.

This paper is organized as follows. Section 2 introduces related works on prediction techniques and grid schedulers where computer selection is supported by metrics. Section 3 introduces the problem addressed by this paper. In Section 4, the Productivity Forecasting System is presented. An explanation of the heuristic used by the system is presented in Section 5. In Section 6 the simulation process used to evaluate our proposal is explained followed by an analysis of the experiments results. Finally, in Section 7 the conclusions and future works are presented.

2 RELATED WORKS

We did not find in the current literature works about a system to forecast a computer productivity based on a heuristic method to improve grid applications performance in EDGs. Nevertheless, our work is based on two important research areas:

[1] `http://www.uncu.edu.ar/`

1. prediction methods to estimate a computer availability and
2. grid schedulers that use information such as computer availability to improve its performance.

A computer status can be calculated from an estimation of its availability or load. Next, works related to both areas are introduced.

Computer availability can be understood as

1. CPU availability and
2. host availability.

CPU availability refers to a computer status where a host is able to run grid applications. On the other hand, host availability denotes a computer status where it is possible to access a host via a grid middleware.

CPU availability has been deeply studied over years. A comprehensive analysis of computer load presented in [6] concluded that CPU load exhibits complex properties such as self-similar and seasonal behavior. Even so, it is possible to predict with a good accuracy short-term CPU load having as input historical data. Encouraged by this result, the author analyzed different linear time series models in [8]. This work demonstrates that computer load on real systems is predictable from past performance using linear time series techniques. However, these models were accurate for short-term (a few seconds ahead) predictions. They also have different computational requirements making some of them more practical than others to perform predictions in real time. Both works fostered the development of different prediction strategies. Real-time systems dynamism required new prediction strategies to deal with real-time applications. The CPU availability prediction model presented in [2] is able to make predictions from real-time measurements provided by a monitoring tool in UNIX systems with a low error percentage. In real-time systems it is important to reduce prediction errors. To this end, a method for error correction of predictions based on regression methods is presented in [5]. The authors propose to build a secondary regression predictor whose task is to predict the signed error of the prediction which was made using the original regression model.

Changes in the CPU load tendency a.k.a. seasonal variance may affect prediction methods accuracy. A prediction strategy to overcome this drawback is presented in [36]. The proposed prediction method is based on the analysis of tendencies in previous CPU load data and similarities in behavior patterns. It uses a polynomial fitting method to improve CPU load prediction accuracy. Another effort to deal with seasonal CPU variance is presented in [26].

Prediction methods have drawn much attention in grid computing systems [12, 32]. One common predictor is the Network Weather Service (NWS) [21, 20]. This system has a module that uses different forecasters to predict a computer performance. Historical data is processed by each forecaster and an accuracy tournament process selects which forecaster will be used. The data collected include: fractions of CPU time available for new processes, TCP connection time, bandwidth and end-to-end round-trip network latency values. The prediction method used by the NWS is able to deliver accurate predictions in many situations including CPU availability [22].

In general, linear models are accurate for short-term predictions but their parameters may vary from computer to computer [9]. For this reason new methods were developed and tested, for instance the Adaptive Neuro Fuzzy Inference System (ANFIS) from the Artificial Intelligence (AI) area. ANFIS is a kind of neural network that is based on a fuzzy inference system that integrates both neural networks and fuzzy logic principles. Its inference system corresponds to a set of fuzzy IFTHEN rules that have learning capabilities to approximate nonlinear functions such as CPU load. A model based on ANFIS via naive Bayes assumptions is presented in [3]. In [19] a fuzzy stochastic technique is used to predict mainframe CPU utilization to help programmers tune its performance.

The K-means clustering algorithm [11] is another method from AI area. It has been used to make predictions of long-term computer availability [14]. This algorithm is used to find a pattern behavior in computer historical data to predict its future availability. In [32] K-means is used to discover subsets of hosts with similar statistical properties within a large-scale distributed system. Classification algorithms [1] from AI can be used to predict a computer status based on its historical performance data. We tested some of these algorithms in a previous work [37]. For instance, the C4.5 algorithm [17] showed an average accuracy up to 75 % predicting a computer status. Additionally, these algorithms are able to calculate long-term predictions in a few seconds.

Host availability prediction is useful when it is not possible to access CPU availability data. Monitoring computers activity is not always feasible because of administrative bureaucracy or security policies. For this reason, we include another technique in this section that may help to predict host availability as alternative to CPU availability. Host availability has been deeply analyzed from traces in different works [29, 4, 10]. These preliminary studies lead to different host availability prediction methods. In [35], a method to identify under-utilized CPUs based on traffic network information is presented.

AI algorithms have also been used in this research area. In contrast to statistical methods, some of these algorithms do not require to build a prior model neither to identify the underlying data distribution. In addition, they are adaptive in nature by updating historical data to generate a prediction. For instance, lazy learning algorithms [24] such as K-Nearest Neighbors and Naive Bayes are suitable for host availability prediction as shown in [15]. The K-means clustering algorithm [11] has also been used in [33] to measure and characterize host availability in large-scale Internet distributed systems such as volunteer computing and grid systems. This strategy identifies computer groups with a correlated availability that exhibits similar time effects. As a result, these groups can improve resource management in volunteer computing systems.

Prediction techniques are useful to improve scheduling system performance. The real-time scheduling advisor (RTSA) [31] is a user-level system that advices how to schedule real-time tasks. The goal of the RTSA is to help client application to meet deadlines and to report when deadlines cannot be met. The resource prediction is based on statistical time series analysis documented in [6, 7]. The system suggests a resource from a set of computers such that a task with nominal runtime t_{nom}, if started now, will be completed in time $(1 + sf)t_{nom}$ or less with a confidence $conf$ where sf is the slack factor. The system response consists of a copy of the request t_{nom}, sf and $conf$ values, the selected host and an estimated task runtime. We did not compare our proposal with this work because the RTSA helps users to select computers where to run their applications whilst PFS aims to improve the grid scheduler computer selection process in EDGs.

A decentralized approach for volunteer computing systems based on resource availability prediction is presented in [16]. Resource availability prediction is computed considering three input factors:

1. resource availability,
2. group availability and
3. current group availability.

The first factor is computed from a historical resource activity register. It represents the probability of individual machines availability at a given time of the week. The second factor quantifies the number of resources available at a given time. Finally, the third factor is calculated based on the number of busy and idle computers. This model reduces the number of job interruptions in a distributed system of non-dedicated desktop computers. Similar to PFS, this work computes a numerical value used by the scheduler to select computers where to run Grid applications. We refer to this work as R.A.P. (Resource Availability Probability) approach and it will be compared with PFS in the experimental process.

In summary, there are many efforts that predict a computer status with a very good accuracy. We propose to take advantage of these methods to forecast a computer productivity. We argue that computer productivity forecasting may help the scheduling system decisions to make a better assignment of applications to computers and, in turn, to reduce grid applications runtime. To this end, we propose to use current prediction methods. Next, we state the problem addressed in this paper.

3 PROBLEM STATEMENT

In EDGs, a grid scheduler has to deal with the fact that users might claim any computer anytime. As a consequence, a grid application can be suspended several times until finishing its execution. A possible solution could be to move a suspended application to an idle computer. This situation could be repeated until finishing the application but it is costly in terms of network usage and processing time. Although many efforts try to predict a computer status with a good accuracy this information is not enough. Let us consider the following simplified example. A grid scheduler has to run a grid application app_1 with an unknown runtime. The grid scheduler requests to a Resource Discovery System a set of computers where to run this applications based on computer capabilities constrains. The scheduler receives a response that includes nine computers with the same hardware and software features. Based on existing works, it is possible to predict computer statuses for a discrete time interval that goes from t_1 to t_{10} as shown in Figure 1. To simplify the example, let us suppose computers will be idle the same units of time to run the application app_1. However, computers will not finish running the application at the same time. For instance, if application runtime was 5 units of time then computer $c1$ would finish at the instant t_5 whilst $c9$ would finish at t_{10}.

	c1	c2	c3	c4	c5	c6	c7	c8	c9
t1	Idle	Busy	Busy	Idle	Idle	Idle	Idle	Idle	Busy
t2	Idle	Idle	Busy	Idle	Idle	Idle	Idle	Busy	Idle
t3	Idle	Idle	Busy	Idle	Idle	Busy	Busy	Idle	Busy
t4	Idle	Idle	Busy	Idle	Busy	Idle	Idle	Busy	Idle
t5	Idle	Idle	Busy	Busy	Idle	Idle	Busy	Idle	Busy
t6	Busy	Idle	Idle	Idle	Busy	Busy	Idle	Busy	Idle
t7	Busy	Busy	Idle	Off	Busy	Idle	Busy	Busy	Busy
t8	Busy	Busy	Idle	Off	Busy	Busy	Busy	Idle	Idle
t9	Busy	Busy	Idle	Off	Busy	Busy	Busy	Busy	Busy
t10	Busy	Busy	Idle	Off	Idle	Busy	Idle	Idle	Idle

Figure 1. Computer status prediction

In this example, if the grid scheduler chooses $c1$, this computer will finish running app_1 faster than other candidate computers. Our proposal is founded in previous discussed works and supported by the following premises:

- A computer productivity can be calculated having as input that computer status changes.
- There is a relationship between a computer productivity and a grid application runtime and it can be calculated as we will show in Section 5.
- This relationship is present in volunteer computing systems because they use idle computer time donated by users.
- It is possible to forecast a computer productivity having as input a prediction of future computer statuses.

We argue that computer selection made by a grid scheduler based on our proposal may lead to improvements in grid applications runtime. To this end, we propose a system to predict a computer productivity based on a prediction of computer statuses. A heuristic method calculates and predicts a computer productivity. The system runs on every computer and its forecast productivity is reported to grid schedulers via a Resource Discovery System. Next, we introduce a detailed explanation of the Productivity Forecasting System.

4 THE PRODUCTIVITY FORECASTING SYSTEM

The PFS is designed to be executed as a software application on every desktop computer. In this way, computers are able to provide information to a grid scheduler about their expected productivity via a Resource Discovery System. The system is composed of a set of modules to monitor a computer activity, to make a prediction of computer statuses and based on this information to forecast a computer productivity. Computer statuses can be calculated using any prediction method. The PFS modules are as follows:

1. Computer Activity Sensor,
2. PFS Interface,
3. Computer Status Predictor and
4. Computer Productivity Forecaster.

Interaction between modules is shown in Figure 2.

The computer activity sensor monitors and records data about a computer activity at a predefined frequency. In general, prediction methods use this information to predict a computer status. The PFS interface is responsible for solving queries about a computer productivity forecasting. These queries are broadcast via a Resource Discovery System. A query specifies a set of parameters required to calculate that productivity. To solve a query, the interface requests the computer status prediction module to predict a sequence of computer statuses. Based on this information, the productivity forecaster module calculates an expected computer productivity that is sent back to the PFS interface. Next, a more detailed explanation of the system modules is introduced.

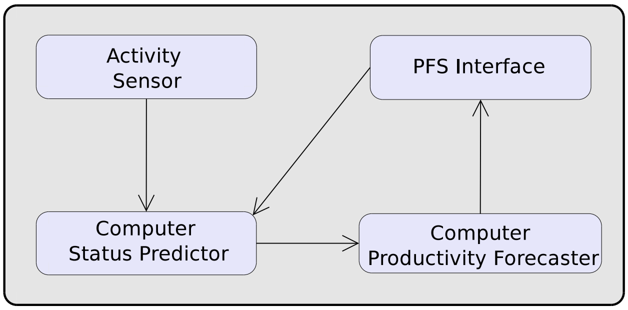


Figure 2. The productivity forecasting system

4.1 Computer Activity Sensor

The computer activity sensor periodically monitors and records data about each computer activity. Every sensor uses the same configuration settings previously defined by a PFS developer. The traces used in our experiments have information monitored with a frequency of 1 minute. In future works, we will analyze whether changes in the sampling frequency affect productivity forecasts.

To keep consistent information, computers clocks are synchronized with a central server using NTP. The sensor records the following data:

1. date and time,
2. percentage of free RAM and
3. CPU load.

This process does not degrade computer performance.

The sensor uses an operating system daemon service. The most popular services are the Service Activity Report for Linux and the Windows Performance Monitor for Windows systems. In general, the storage space required to record the monitored information is about 50 KBytes per day. The oldest records can be erased to limit the volume of the stored information.

4.2 PFS Interface

The system interface is responsible for managing each system module. This module takes as input a query that provides a set of configuration parameters required by the system. Some of them include:

1. computer status prediction method to be used,
2. parameters required by that method,
3. period of time to be included in the prediction and
4. units of time considered in the prediction.

When the interface receives a query, it triggers a computer status prediction process. Based on this information, the computer productivity forecaster module calculates the expected productivity of that computer and it is sent back to the interface. As mentioned before, it is expected that the PFS results are delivered to grid scheduler via a Resource Discovery System. Interaction details between the system interface module and a Resource Discovery System are part of future works.

4.3 Computer Status Predictor

The computer status predictor module is responsible for estimating future computer statuses for a discrete period of time. Computer statuses include:

1. idle,
2. busy and
3. off.

We use this simplified scheme because our interest is to predict if a computer is idle or not to execute a grid application. A computer status is calculated based on information provided by the sensor module. The prediction process is triggered by the system interface module.

The module results are represented by an array called Computer Status Array. Predictions are made using any method such as classification algorithms from AI, the Network Weather Service, time series analysis (among others mentioned in Section 2). An example is shown in Table 1 where the prediction time interval includes 12 hours, from 09:00 to 21:00 at a frequency of ten minutes.

09:00	09:10	09:20	09:30	09:40	09:50	...	20:40	20:50	21:00
idle	idle	idle	idle	busy	busy	...	off	off	off

Table 1. Computer Status Array example

A computer status array requires additional information for further analysis made by the Forecaster module and a grid scheduler. For this reason, every array has associated a Computer Status Array Header. It provides the following information:

1. computer time zone,
2. date and time when the prediction was computed,
3. start date,
4. end date,
5. sampling frequency,
6. prediction method and
7. prediction method accuracy.

Table 2 shows an example of a header associated to the computer status array shown before.

Time zone	(UTC-03:00) Buenos Aires
Estimation date	02/11/2011 08:59:00
Start date	02/11/2011 09:00:00
End date	02/11/2010 21:00:00
Frequency	600
Prediction method	C4.5
Accuracy	80 %

Table 2. Computer Status Array Header example

4.4 Computer Productivity Forecaster

The computer productivity forecaster module is the component responsible for calculating the expected productivity for a computer. As mentioned before, we argue that changes in a computer statuses affect a computer productivity. Furthermore, this productivity has a relationship to a grid application runtime. An important challenge is how to calculate and predict that productivity. To this end, we developed a heuristic method which is introduced in the next section.

5 HEURISTIC METHOD FOR PRODUCTIVITY FORECASTING

The main goal of the proposed heuristic method is to identify from a set of computers which one may finish running a grid application faster. We developed a heuristic method based on the thesis that there is an important relationship between computer status changes and grid applications runtime. To identify that relationship we monitored desktop computers activity from an university campus. Afterwards, we analyzed why a set of identical computers might finish running the same application at different times.

From that analysis, we identified that an application runtime is affected by the following factors:

1. How long a computer is idle to run a grid application: a computer may be idle many units of time but those units can be spread over an extended period of time which might delay the execution of a grid application.
2. When a computer will be able to start running a grid application: a computer may remain idle for a large period of time. Nevertheless, the difference between the moment when an application should start running and the moment when that computer is ready to start that application could be significant.
3. How many times a grid application is suspended by users: a computer may start running an application immediately. However, that computer could be interrupted several times introducing an important delay in that application runtime.
4. How long a grid application is suspended: a computer can be interrupted a few times but those interruptions may last a long period of time.

Considering these observations, we conclude that all factors mentioned before affect a grid application runtime. Those factors should be taken into account to estimate which computers will finish running an application earlier than other computers with the same computational power. To this end, we defined a set of metrics to calculate what we called in this work "*computer productivity*". The metrics are the following:

- *Statuscount*: the number of instances of certain status in the computer status array.
- *Clustering*: the number of clusters of a selected status in the computer status array.

- *Cohesion*: this value represents how spare are clusters of the status being analyzed in the computer status array. This metric is calculated according to the following formula:
$$Cohesion = \frac{Statuscount}{Clustering}.$$
- *Statusscope*: this value counts the number of instants t_i from the first to the last status occurrence.
- *Productivity*: this value provides a measure that aims to represent a computer productivity capability. This metric relates the factors that may affect a grid application runtime. Computers that show higher values are more likely to run applications faster. This metric is computed as follows:
$$Productivity = \frac{Cohesion}{Statusscope}.$$

According to our traces analysis, computers with a productivity that tends to 1 finish running application earlier than computers with a lower productivity value. This experimental validation requires a more detailed study in future works. The analysis of past computer statuses enables the heuristic method design. Consider that the proposed heuristic method calculates a set of metrics based on a sequence of computer statuses. Then, it is possible to estimate the future productivity of a computer. To predict those computer statuses there are a broad number of techniques described in Section 2. For this reason, we propose to forecast a computer productivity based on a prediction of computer statuses.

5.1 Productivity Forecasting System Integration with an EDG

As mentioned before, the system is designed to run on each computer that participates in an EDG. The main goal is to provide the information calculated by our proposal to a grid scheduler. To this end, a grid scheduler sends a request of computers to a Resource Discovery System. This system broadcasts a query to participant computers. Each computer forecasts its productivity based on a set of parameters. Then, a computer will send a response back only if that computer fulfills two requirements:

1. computational power and
2. waiting threshold.

The first parameter describes software and hardware computer features required by grid applications to run. The second parameter is used to filter out those computers that might take a long time to start running a grid application. For instance, a grid application app_1 should start running at an instant of time t_1 and its startup should not be delayed by certain units of time. If a computer fulfills both requirements then it sends back a response to the Resource Discovery System. Afterwards, a ranking of computers ordered by their expected productivity is sent back to a grid scheduler. Finally, one or more applications are sent by the grid scheduler to the selected computers to be executed. Interaction between the components mentioned before is presented in Figure 3.

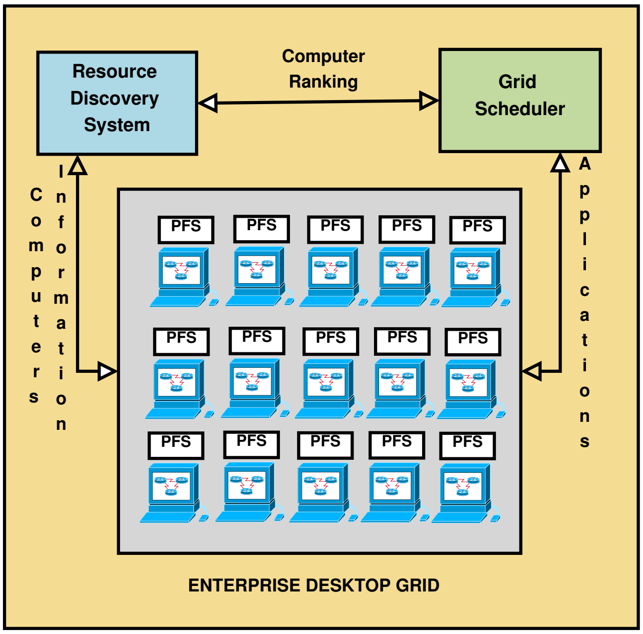
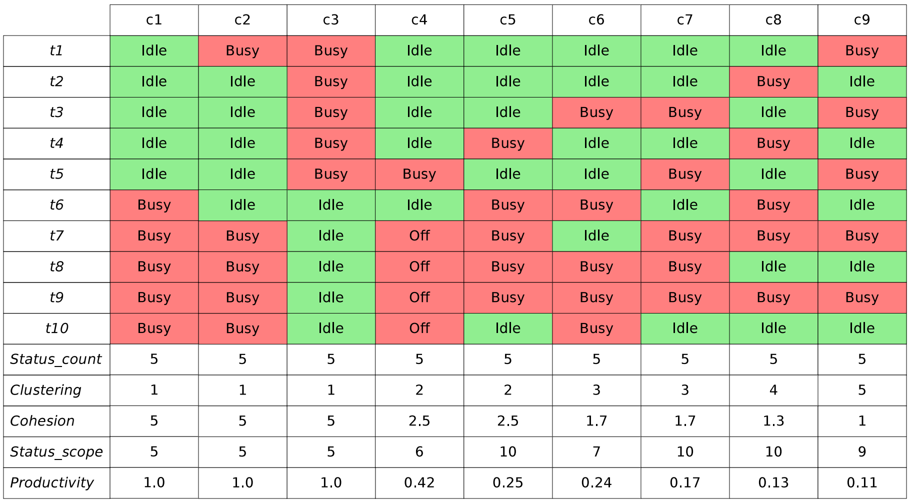


Figure 3. Grid Scheduler, RDS and PFS interaction on an EDG

In Section 3, we presented a simplified example of the problem addressed by our proposal. In that example, a grid scheduler requires a desktop computer to run a grid application. Application runtime is unknown and the start up of this application should not be delayed by more than 5 units of time. The value for the parameter waiting threshold is set by users. The Resource Discovery System broadcasts a query with details about computers requirements. In the example shown in Figure 4 nine computers fulfill computational power constrains. Nevertheless, since the waiting threshold for the example is 5 units, computer $c2$ does not reply back to the Resource Discovery System.

Based on the responses received from the remaining 8 computers, a ranking is created ordered by their expected productivity as shown in Figure 4.

To simplify the explanation of the heuristic method, let us consider that all computers are idle the same units of time. If a grid application runtime was 5 units of time then computers with higher expected productivity would run an application faster than others. For instance, $c1$ and $c3$ have an expected productivity of 1. According to Figure 4 those computers would finish running that application



	c1	c2	c3	c4	c5	c6	c7	c8	c9
t1	Idle	Busy	Busy	Idle	Idle	Idle	Idle	Idle	Busy
t2	Idle	Idle	Busy	Idle	Idle	Idle	Idle	Busy	Idle
t3	Idle	Idle	Busy	Idle	Idle	Busy	Busy	Idle	Busy
t4	Idle	Idle	Busy	Idle	Busy	Idle	Idle	Busy	Idle
t5	Idle	Idle	Busy	Busy	Idle	Idle	Busy	Idle	Busy
t6	Busy	Idle	Idle	Idle	Busy	Busy	Idle	Busy	Idle
t7	Busy	Busy	Idle	Off	Busy	Idle	Busy	Busy	Busy
t8	Busy	Busy	Idle	Off	Busy	Busy	Busy	Idle	Idle
t9	Busy	Busy	Idle	Off	Busy	Busy	Busy	Busy	Busy
t10	Busy	Busy	Idle	Off	Idle	Busy	Idle	Idle	Idle
Status_count	5	5	5	5	5	5	5	5	5
Clustering	1	1	1	2	2	3	3	4	5
Cohesion	5	5	5	2.5	2.5	1.7	1.7	1.3	1
Status_scope	5	5	5	6	10	7	10	10	9
Productivity	1.0	1.0	1.0	0.42	0.25	0.24	0.17	0.13	0.11

Figure 4. Example of a set of computers and their Productivity Forecast

faster than the rest of the other computers. Computers at the top of the ranking would finish running an application faster than those at the list bottom. Despite the simplicity of this example, experiments revealed that our proposal runs grid applications faster than other approaches. However, the proposed metrics require a more detailed analysis.

6 EXPERIMENTS AND RESULTS

To validate our proposal, we evaluated performance improvements achieved by a grid scheduler supported by the Productivity Forecasting System. To this end, we developed a simulator that runs three different Grid scheduler configurations. The first one uses information about computer productivity provided by the Productivity Forecasting System to select computers where to run grid applications. The second grid scheduler configuration selects computers randomly. Finally, the third one selects computers based on a resource availability probability (R.A.P.) metric [16].

To calculate a computer status prediction, we used a classification algorithm called C4.5 a.k.a. J48 [17]. This algorithm was the most accurate from a set of classification algorithms tested before running our experiments. Results of algorithm testing are introduced in Section 6.2. The algorithm C4.5 was used to validate a preliminary version of our proposal [37]. Computer status prediction using classification algorithms requires an extensive explanation that will be presented in future works.

For the sake of simplicity, we have considered the following assumptions:

- Application execution time does not depend on computer hardware and software features but on idle computer time.

- Computers run applications whenever they are idle according to computer activity traces.
- If a computer is turned off and a grid application has not finished, it will be resumed when the computer is on and its status is idle again.

The simulator ran 157 500 grid applications with different runtime, between 600 and 5 400 seconds in the time interval from 08:00 and 15:00. In this period of time most of computers are being used by teachers and students. Therefore, if we run grid applications they will be suspended by those computer users several times. For this reason this time interval is appropriate to test our system.

A variable number of grid applications were launched simultaneously in half an hour periods of time. This number was calculated as a percentage of participant computers. Thus, different workloads were tested in a range from 1 % to 100 %. In this way, the proposed system was stressed because when workloads grow the number of computers to be selected diminishes.

We split the experimental process into four stages:

1. desktop computer traces analysis,
2. classification algorithm accuracy analysis,
3. grid applications instances definition and
4. grid scheduler performance evaluation.

Experiments use a set of traces from desktop computers at UNCuyo University. In a first stage, we analyzed if the collected information from desktop computers was suitable to run our experiments. Then, we evaluated different classification algorithms accuracy to select the most accurate algorithm to be included in the experimental process. Afterwards, we created a significant number of grid applications with different configurations. Finally, we ran the generated applications for each grid scheduler configuration and calculated the average application runtime. Next, we present each phase in details.

6.1 Desktop Computer Traces

For a period of thirty five business days, twenty computers were monitored. On each computer, a software application recorded the following data every minute:

1. date and time,
2. CPU load and
3. free RAM.

This information was integrated into a database for a further analysis.

Desktop computers are placed at laboratories used for teaching purposes. Students use computers for software development and operating system management. In general, computers stay on from Monday to Friday between 07:30 to 18:00.

We preprocessed the computer traces to analyze them as follows. For each instant of time, we set a computer status (idle, busy or off) according to the CPU load and free RAM. For instance, a computer status is idle whenever the CPU load is zero and the RAM is under a predefined threshold. Then, the percentage of idle computers was calculated for the period of time included in the experiments. The result is shown in Figure 5.

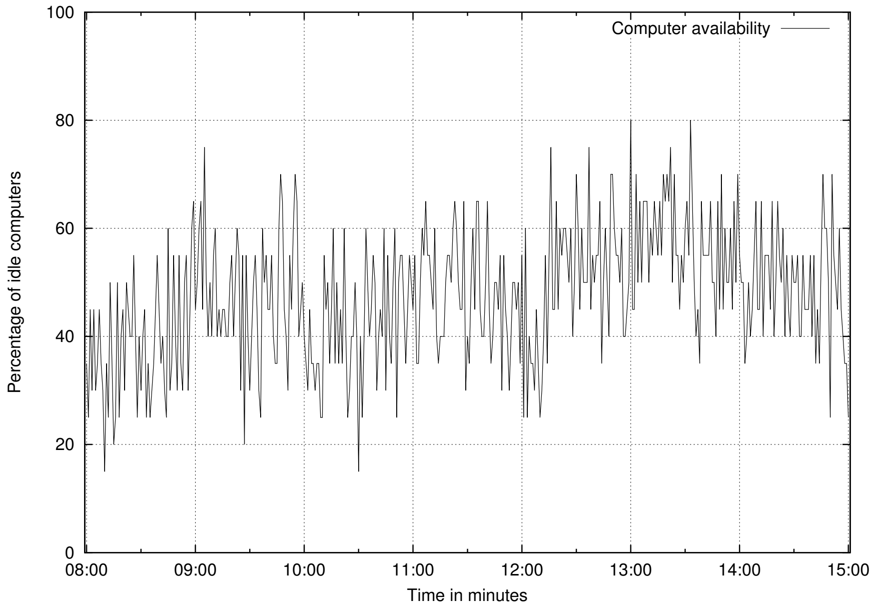


Figure 5. Percentage of idle computers from 08:00 to 15:00

It can be observed that the percentage of idle computers changes every minute. Those computers that change their status from idle to busy suspend any grid activity. As a consequence, selecting the right computers where to run grid applications is important to make an optimal usage of idle computers time. We conclude that the collected information about computer activity is appropriate to run our experiments.

6.2 Classification Algorithms Accuracy

The proposed system can use any prediction technique to estimate a computer status for an instant of time. As mentioned before, we proposed the usage of classification algorithms from the artificial intelligence area. For this reason, we selected and evaluated the prediction accuracy of different algorithms.

The Weka [1] framework was used to evaluate classification algorithms accuracy. The framework provides a module to test a set of predefined classification algorithms. To test an algorithm, a user defines a set of configuration paramenter values and

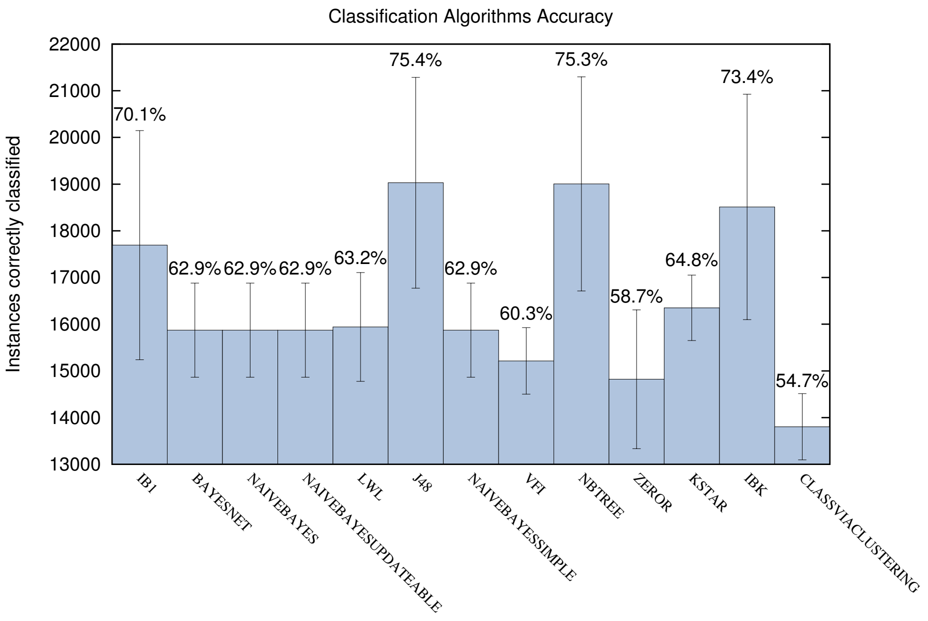


Figure 6. Classification algorithms accuracy for computer status prediction

provides a data set to be used in the process. In this case, we used monitoring information from computers.

The tested algorithms were the following: IB1, BayesNet, Naive Bayes, Naive Bayes updateable, LWL, J48, Naive Bayes Simple, VFI, NBTree, Zeror, KStar, IBK and classification via clustering. For a detailed description of these algorithms please refer to [38]. We evaluated each algorithm using every available data set. Afterwards, we calculated the average accuracy and standard deviation. Results are shown in Figure 6. Bars represent the average number of instances correctly classified and on top of each bar average accuracy and standard deviation for each algorithm are shown. The algorithm called *J48* achieved the highest percentage of accuracy. For this reason, it was used by the computer status predictor module in the simulation process.

6.3 Grid Application Instances

Different grid applications instances were defined before starting running the simulator. In this way, it was possible to create different scenarios to evaluate our proposal. Since our proposal selects computers that are expected to run the application faster, the proposed system will be stressed if the number of options is reduced. For this reason, we created sets of applications to be launched simultaneously. The number of applications per set was calculated based on the number of participant computers creating workloads from 1 % to 100 %.

Application sets were configured to be scheduled in a period of time from 08:00 to 15:00 split into 15 intervals of half an hour. For each set, applications startup time was chosen randomly within each time interval. Application duration ranges from 600 and 5 400 seconds and was generated randomly. This process was repeated a configurable number of times represented by the variable *num_iteration*.

The number of grid applications instances created was calculated according to Equation (1):

$$total\_app = num\_app * num\_intervals * num\_iteration \tag{1}$$

The number of grid application *num_app* at every time interval was calculated as shown in Equation (2):

$$num\_app = \sum_{i=0.1}^{1} n * i \tag{2}$$

where $i = i + 0.05$ and n is the number of participant computers.

The simulator used the following configuration. For each one of the 15 time intervals, a number of application instances *num_app* is generated according to Equation (2), which in this case is 210. Since this process was repeated 50 times the number of application instances created was 157 500.

6.4 Grid Scheduler Performance Evaluation

Experiments are designed to assess how our proposal can improve grid scheduling systems performance. To this end, the metric to be considered is the average runtime achieved by different grid scheduler configurations. In this stage, the simulator takes as input the grid application sets and the computer traces.

Once the simulator processed all grid application instances the average runtime was calculated. Results include the average grid application runtime achieved by fully-dedicated computers to measure the improvement achieved in comparison with other strategies. We present results performance achieved under different workloads.

Figures 7 to 10 show the average application runtime for different workloads. For the sake of simplicity, we present results achieved for workloads of 5 %, 25 %, 75 % and 100 %. Each graph represents results calculated for the period of time from 08:00 to 15:00.

In most cases, we observe that the runtime standard deviation for the other strategies is significantly greater than our proposal. The average runtime for the Productivity Forecasting System is not significantly degraded by workload variations. A slight difference is observed at 11:00 with a workload of 10 % in Figure 7 where random selection outperforms the other strategies. A possible explanation is that the number of idle computers decreases between 10:00 and 11:00 according to the percentage of idle computers shown in Figure 5.

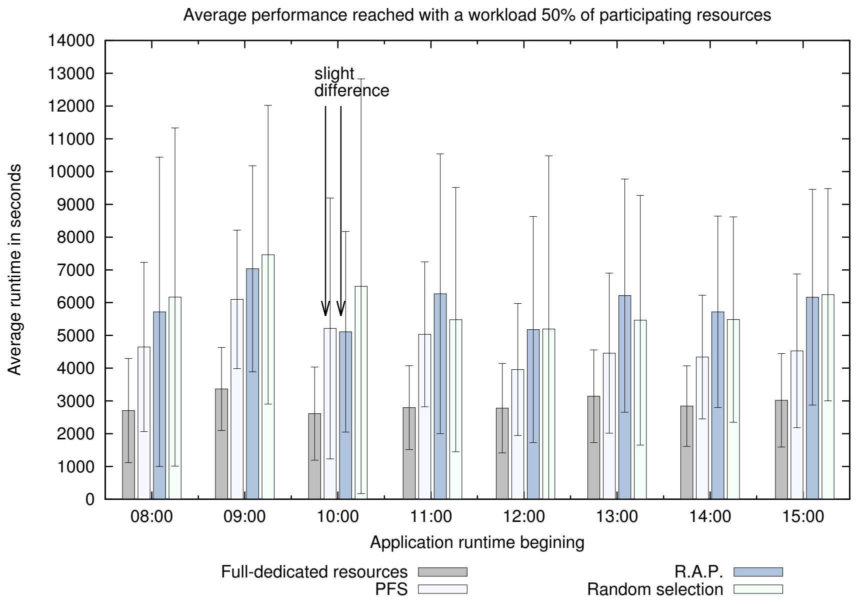


Figure 7. Average runtime for a workload of 5 %

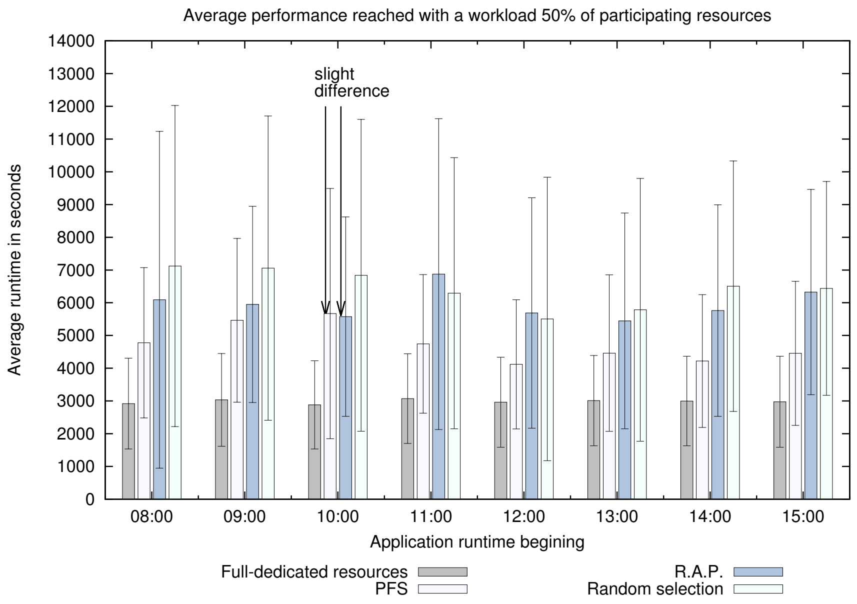


Figure 8. Average runtime for a workload of 25 %

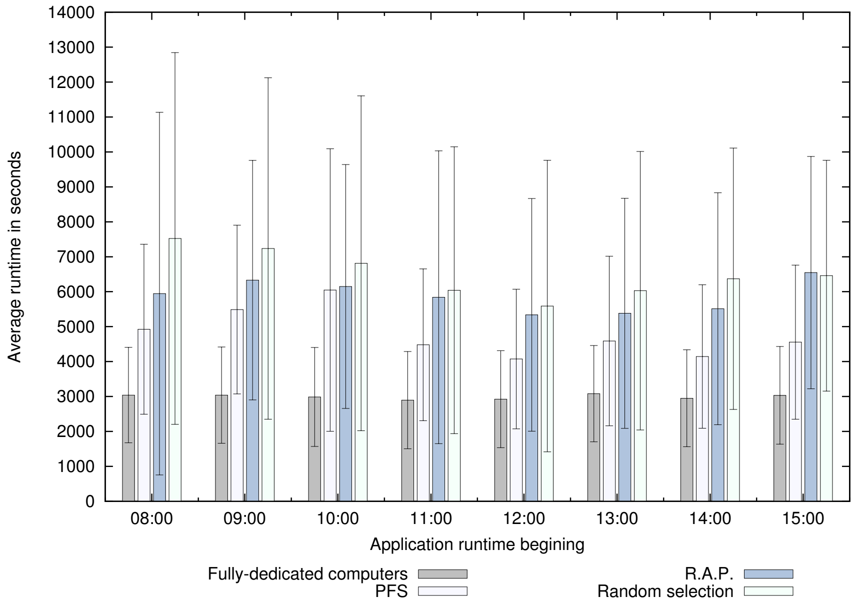


Figure 9. Average runtime for a workload of 75 %

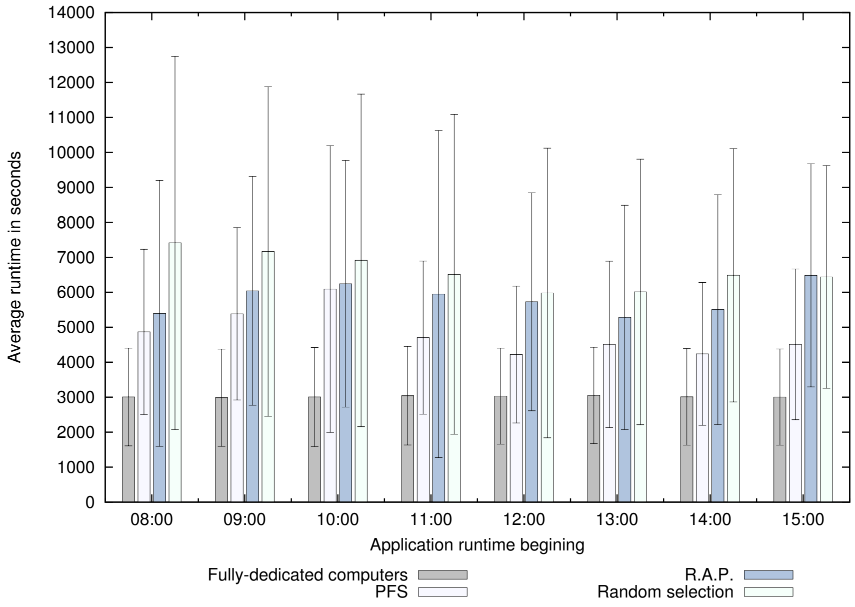


Figure 10. Average runtime for a workload of 100 %

In summary, PFS outperformed other strategies in almost all cases. Additionally, the runtime standard deviation for our proposal is smaller than other approaches. In some cases, PFS ran grid applications up to 80 % faster than other strategies. To calculate this value, we considered the runtime achieved by the applications simulated. Then, we calculated the percentage of improvement achieved by our proposal in comparison with the best performance of other strategies. For instance, the runtime for an application app_1 was 1 000, 5 000 and 5 500 seconds using PFS, RAP and random selection, respectively. For this example, we compared PFS and RAP runtime where our proposal ran app_1 80 % faster than other approaches.

7 CONCLUSIONS AND FUTURE WORKS

In this paper we proposed a computer productivity forecasting system to improve grid application runtime in EDGs. The system calculates and predicts a computer productivity. We are interested in computer productivity because it significantly influences grid application runtime in volunteer computing systems. For this reason, our proposal predicts a computer productivity based on a prediction of a sequence of computer statuses. Experimental results show that a grid scheduler can improve application execution performance if it chooses computers based on their expected future productivity.

To validate our proposal, we developed a simulator to run grid applications in EDGs environment. The simulator has a grid scheduler that uses three computer selection strategies:

1. selection based on our proposal,
2. selection based on R.A.P. and
3. random selection.

We used computer activity traces from desktop computer placed at a university campus. In the first stage, we analyzed computer traces to verify that computer usage patterns are useful for simulating an EDG environment. The collected information showed that the average number of idle computers varies over time in such a way that is suitable for running our experiments. In the second stage, 157 500 grid application instances were created with an execution time that varied from 600 to 5 400 seconds. Applications were launched simultaneously creating different workloads to stress our proposal by reducing the number of computers to be selected. We compared the average runtime achieved by our proposal against the other computer selection strategies.

Results show that a grid scheduler based on PFS runs applications up to 80 % faster than the other strategies. We conclude that computer status changes do affect grid applications runtime. We tested a heuristic method to calculate a computer productivity based on a set of metrics that represent computer status changes. On the other hand, computer status prediction plays an important role in the Productivity Forecasting System and it deserves a deeper analysis.

In future works, we will analyze in depth issues such as:

1. prediction techniques,
2. the heuristic method used by our proposal,
3. PFS performance on different computing environments and
4. PFS and resource discovery system integration.

In contrast to other works, our experiments used classification algorithms to predict a computer status based on historical information. We will perform a more detailed assessment of classification algorithms applied to status predictions; for instance, the amount of monitored information over time, the number of fields considered by the algorithms, changes in computer usage that may alter classification algorithms performance. On the other hand, we will accomplish a more detailed comparison of classification algorithms and other prediction techniques.

The heuristic method used by our proposal is based on the calculation of a set metrics. We will carry out a more detailed analysis of this metrics and its relationship to grid applications performance. Results may lead to changes in the heuristic method to improve the performance of our proposal.

Acknowledgment

This research is supported by the National Agency for Scientific and Technological Promotion (ANPCyT) through project No. PAE-PICT 2312. This study is also founded by the Argentine Ministry of Science and Technology (MINCyT). The first author wants to thank the ANPCyT for the granted fellowship. The financial support provided by the SeCTyP-UNCuyo through projects No. 06B/194 and No. 06B/253 is gratefully acknowledged.

REFERENCES

[1] Weka Machine Learning Project web site. Availaible on: `http://www.cs.waikato.ac.nz`.

[2] Beltran, M.—Guzman, A.—Bosque, J. L.: A New CPU Availability Prediction Model for Time-Shared Systems. IEEE Transactions on Computers, Vol. 57, 2008, No. 7, pp. 865–875.

[3] Bey, K.—Benhammadi, F.—Mokhtari,A — Gessoum Z.: Mixture of ANFIS Systems for CPU Load Prediction in Metacomputing Environment. Future Generation Computer Systems, Vol. 26, 2010, No. 7, pp. 1003–1011.

[4] Bolosky, W.—Douceur, J.—Ely, D.—Theimer, M.: Feasibility of a Serverless Distributed File System Deployed on an Existing set of Desktop PCs. SIGMETRICS Performance Evaluation Review, Vol. 28, 2000, No. 1, pp. 34–43.

[5] Bosnic, Z.—Kononenko, I: Correction of Regression Predictions Using the Secondary Learner on the Sensitivity Analysis Outputs. Computing and Informatics, Vol. 29, 2010, No. 6, pp. 929–946.

[6] Dinda, P.: The Statistical Properties of Host Load. Scientific Programming, Vol. 7, 1999, No. 3, pp. 211–229.

[7] Dinda, P.: Online Prediction of the Running Time of Tasks. Cluster Computing, Vol. 5, 2002, No. 3, pp. 225–236.

[8] Dinda, P.: Host Load Prediction Using Linear Models. Cluster Computing, Vol. 3, 2000, No. 4, pp. 265–280.

[9] Dong, F.—Luo, J.—Song, A.—Cao, J.—Shen, J.: An Effective Data Aggregation Based Adaptive Long Term CPU Load Prediction Mechanism on Computational Grid. Future Generation Computer Systems, Vol. 28, 2012, No. 7, pp. 1030–1044.

[10] Douceur, J.: Is Remote Host Availability Governed by a Universal Law? SIGMETRICS Performance Evaluation Review, Vol. 31, 2003, No. 3, pp. 25–29.

[11] Hartigan, J.—Wong, M.: A K-Means Clustering Algorithm. Applied Statistics, Vol. 28, 1979, No. 1, pp. 100–108.

[12] Jarvis, S.—Spooner, D.—Lim Choi Keung, H.—Cao, J.—Saini, S.—Nudd, G.: Performance Prediction and Its Use in Parallel and Distributed Computing Systems. Future Generation Computer Systems, Vol. 22, 2006, No. 7, pp. 745–754.

[13] Kondo, D.—Fedak, G.—Cappello, F.—Chien, A.—Casanova, H.: Characterizing Resource Availability in Enterprise Desktop Grids. Future Generation Computer Systems, Vol. 23, 2007, No. 7, pp. 888–903.

[14] Lázaro, D.—Kondo, D.—Marquès, J.: Long-Term Availability Prediction for Groups of Volunteer Resources. Journal of Parallel and Distributed Computing, Vol. 72, 2012, No. 2, pp. 281–296.

[15] Rahman, M.—Hassan, M.—Buyya R.: Jaccard Index Based Availability Prediction in Enterprise Grids. Procedia Computer Science, Vol. 1, 2010, No. 1, pp. 2707–2716.

[16] Ramachandran, K.—Lutfiyya, H.—Perry, M.: Decentralized Approach to Resource Availability Prediction Using Group Availability in a P2P Desktop Grid. Future Generation Computer Systems, Vol. 28, 2012, No. 6, pp. 854–860.

[17] Salzberg, S.: C4.5: Programs for Machine Learning. Machine Learning, Vol. 16, 1994, No. 3, pp. 235–240.

[18] Trunfio, P.—Talia, D.—Papadakis, H.—Fragopoulou, P.—Mordacchini, M.—Pennanen,M.—Popov, K.—Vlassov, V.—Haridi, S.: Peer-to-Peer Resource Discovery in Grids: Models and Systems. Future Generation Computer Systems, Vol. 23, 2007, No. 7, pp. 864–878.

[19] Wang, Y. F.—Hsu, M. H.—Chuang, Y. L.: Predicting CPU Utilization by Fuzzy Stochastic Prediction. Computing and Informatics, Vol. 20, 2001, No. 1, pp. 20–28.

[20] Wolski, R.: Experiences with Predicting Resource Performance On-Line in Computational Grid Settings. SIGMETRICS Performance Evaluation Review, Vol. 30, 2003, No. 4, pp. 41–49.

[21] Wolski, R.—Spring, N.—Hayes, J.: The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. Future Generation Computer Systems, Vol. 15, 1999, No. 5, pp. 757–768.

[22] Wolski R.: Predicting CPU Availability on the Computational Grid Using the Network Weather Service. Parallel Processing Letters, Vol. 9, 1999, No. 2, pp. 227–241.

[23] Xhafa, F.—Abraham, A.: Computational Models and Heuristic Methods for Grid Scheduling Problems. Future Generation Computer Systems, Vol. 26, 2010, No. 4, pp. 608–621.

[24] Simoudis, E.—Aha, D. eds.: Special Issue on Lazy Learning. Artificial Intelligence Review, Vol. 11, 1997, No. 1-5.

[25] Aloisio, G.—Cafaro, M.—Lezzi, D.: The Desktop Grid Environment Enabler. Computing and Informatics, Vol. 21, 2002, No. 4, pp. 333–345.

[26] Akioka, S.—Muraoka, Y.: Extended Forecast of CPU and Network Load on Computational Grid. In: Cluster Computing and the Grid, IEEE International Symposium on Cluster Computing and Grid (CCGrid 2004), Illinois, April 2004, pp. 765–772.

[27] Anderson, D. P.: BOINC: A System for Public-Resource Computing and Storage. In: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID '04), Pittsburgh, November 2004, pp. 4–10.

[28] Anderson, D. P.—Fedak, G.: The Computational and Storage Potential of Volunteer Computing. In: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID '06), Singapore, May 2006, pp. 73–80.

[29] Bhagwan, R.—Savage, S.—Voelker, G.: Understanding Availability. In: Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), California, February 2003, pp. 256–267.

[30] Coi, S. J.—Kim, H. S.—Byun, E. J.—Baik, M. S.—Kim, S. S.—Park, C. Y.—Hwang, C. S.: Characterizing and Classifying Desktop Grid. In: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGRID '07), Rio de Janeiro, October 2007, pp. 743–748.

[31] Dinda, P.: A Prediction-Based Real-Time Scheduling Advisor. In: Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2002), Florida, April 2002, pp. 10–17.

[32] Javadi, B.—Kondo, D.—Vincent, J.-M.—Anderson, D. P.: Mining for Statistical Models of Availability in Large-Scale Distributed Systems: An Empirical Study of SETI@home. In: IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS '09), London, September 2009, pp. 1–10.

[33] Kondo, D.—Andrzejak, A.—Anderson, D. P.: On Correlated Availability in Internet-Distributed Systems. In: 9th IEEE/ACM International Conference on Grid Computing 2008, Japan, October 2008, pp. 276–283.

[34] Liang, J.—Nahrstedt, K.—Zhou, Y.: Adaptive Multi-Resource Prediction in Distributed Resource Sharing Environment. In: IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2004), Chicago, April 2004, pp. 293–300.

[35] Watkins, L.—Beyah, R.—Corbett, C.: Passive Identification of Under-Utilized CPUs in High Performance Cluster Grid Networks. In: IEEE International Conference on Communications (ICC '08), Beijing, May 2008, pp. 408–413.

[36] Zhou, Y.—Sun, W.—Inoguchi, Y.: CPU Load Predictions on the Computational Grid. In: Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID '06), Singapore, May 2006, pp. 321–326.

[37] Salinas, S.—García Garino, C.—Zunino, A.: An Architecture for Resource Behavior Prediction to Improve Scheduling System Performance on Enterprise Desktop Grids. In: F. V. Cipolla-Ficarra, A. Kratky, K. Veltman, M. Brie, J. Huang, E. Nicol, G. Mori, and M. Cipolla-Ficarra (Eds.): Second International Conference on Advances in New Technologies, Interactive Interfaces and Communicatibility (ADNTIIC '11), Argentina, December 2011, pp. 187–197.

[38] Witten, I. H.—Frank, E.: Data Mining. Practical Machine Learning Tools and Techniques. Elsevier Press 2005.

[39] Li, C.—Li, L.: A Distributed Iterative Algorithm for Optimal Scheduling in Grid Computing. Computing and Informatics, Vol. 26, 2007, No. 6, pp. 605–626.

[40] Santiago, A.—Yuste, A.—Exposito, J.—Galán, S.—Prado, R.: A Multi-Criteria Meta-Fuzzy-Scheduler for Independent Tasks in Grid Computing. Computing and Informatics, Vol. 30, 2011, No. 6, pp. 1201–1223.

[41] Jogalekar, P.—Woodside, M.: Evaluating the Scalability of Distributed Systems. IEEE Transactions on Parallel and Distributed Systems, Vol. 11, 2000, No. 6, pp. 589–603.

Sergio Ariel Salinas is a Ph. D. student in computer science. He graduated in information systems engineering at UTN-FRM University, Mendoza, Argentina in 2004. He is an Adjunct Professor at Universidad Nacional de Cuyo and member of the ITIC Research Institute. His research areas include grid computing, peer-to-peer networks and resource discovery systems.

Carlos García Garino graduated in engineering at University of Buenos Aires, Argentina in 1978 and received the Ph.D. degree from Universidad Politécnica de Cataluña, Barcelona, Spain in 1993. Currently he is Full Professor at the School of Engineering and Head of the ITIC Research Institute, Universidad Nacional de Cuyo, Argentina. His research interests include computational mechanics, computer networks, and distributed computing. He has more than 50 papers published in scientific journals and proceedings of international conferences carried out in Argentina, Brazil, Chile, Canada, Spain, France, Portugal, Belgium and Japan.

Alejandro Zunino received his Ph. D. degree in computer science from The National University of the Center of the Buenos Aires Province (UNICEN), in 2003, and his M. Sc. in systems engineering in 2000. He is a full Adjunct Professor at UNICEN and member of the ISISTAN and the CONICET. His research areas include grid computing, service-oriented computing and mobile computing.