Scheduling Optimization based on Resources State Prediction in Large Scale Distributed Systems

Florin Pop [*] **, Valentin Cristea** [*]

[*] *Faculty of Automatics and Computer Science, University "Politehnica" of Bucharest, Romania, Emails: florin.pop@cs.pub.ro, valentin.cristea@cs.pub.ro*

Abstract: This paper presents an approach for the prediction-based optimization of meta-scheduling in Large Scale Distributed Systems. Several methods are analyzed for resource state prediction to be used in meta-scheduling. Because of the different levels and fluctuations of the performance due to contention caused by competitive applications, schedulers must be able to predict the deliverable performance that an application will be able to obtain when it eventually runs. Time series predictions of the resource status of distributed systems resources, such as CPU or free memory, are considered in order to improve the system availability. The prediction model is based on actual parameter values and historical information, both provided by the MonALISA monitoring system and its extensions: repository and ApMon. The prediction system architecture is extensible, in the sense that other monitoring parameters can be easily added and new prediction models can be included. The predictions of resources are used in meta-scheduling for different types of tasks, especially tasks with dependencies that have an associated communication cost. Based on the dynamic resource information, which sometimes need to be predicted, the scheduler can choose the combination of resources from the available resource pool that is expected to maximize performance for the application and use the resources in an efficient way. The significant improvements obtained for scheduling optimization, with an immediate effect on load balancing and resource utilization are presented.

Keywords: Grid Scheduling, Prediction, Neural Networks, Mon*ALISA*

1. INTRODUCTION

Distributed systems grew rapidly in the past few years and now we consider Large Scale Distributed Systems (LSDS). This was due to the increasing applicability and number of users. As a consequence, LSDS computing became an active subject of the research in distributed systems. In Computational Grid the performance and evaluation are critical components leading to a better scheduling of the jobs, detection of anomalies and also recovery from anomalous behavior, ensuring that the negotiated QoS between the users and providers of resources is met. Finding an effective application performance prediction method for Grid environments constitutes one of the particularly important but still open research problems. The ultimate goal of the application performance analysis and prediction is always answering the question of how particular application will perform in the particular environment. The parallel application runtime prediction is thus the function of used hardware, particular software environment, and the application characteristics. One of the most important aims of LSDS computing is balancing and optimizing the resource utilization. To accomplish these goals, distributed systems use different strategies and several software components, such as management tools, schedulers and monitoring tools.

This paper is based on (1) and proposes a method for predicting optimizing scheduling process using resources' states in LSDS environments. We consider time series predictions of the status of distributed systems resources such as CPU load or available memory. The predictions are based on historical information provided by monitoring systems. The predictions are primarily used in task scheduling, but they can also be used in real-time gathering information about resources. Multiple prediction algorithms have been analyzed and evaluated, which are based on classical mathematical methods for investigating time series data. Based on the real-time measured values, each of these methods is aiming to predict future values. Multi step-ahead prediction is achieved with an algorithm based on a neural network. By using predictions, we can estimate resources' states. The states of resources in distributed systems are used in task scheduling with the purpose to satisfy every task requirements on the one hand, and to

[⋆] The research presented in this paper is supported by national project: "SORMSYS - Resource Management Optimization in Self-Organizing Large Scale Distributes Systems", Project CNCSIS-PN-II-RU-PD ID: 201 (Contract No. 5/28.07.2010). The research is co-founded by national project "DEPSYS - Models and Techniques for ensuring reliability, safety, availability and security of Large Scale Distributes Systems", Project "CNCSIS-IDEI" ID: 1710 (618/15.01.2009). The research was co-founded by the Sectorial Operational Programme Human Resources Development 2007-2013 of the Romanian Ministry of Labour, Family and Social Protection through the Financial Agreement POSDRU/89/1.5/S/62557.

improve the use of available resources, on the other. For tasks with dependencies that are submitted in bulks, a dynamic scheduling approach can be applied but with a small penalty of increasing the response time for these tasks. The resource state predictions can be used in a static scheduling algorithm, with an immediate effect on resource load balancing.

The assumption about the hardware and applications, techniques used to acquire the data and the methods of predicting the behavior are the three most important factors determining the accuracy and the scope of utilization of the particular prediction technique. On the other hand, application runtime information is a fundamental component in application and for the Grid scheduler. Prediction of application's behavior may be used to evaluate with accuracy and eventually rank different resource sets according to their likely performance in order to select the best set of target resources.

This paper is structured as follows: in section two we review related work. In the third section we describe the proposed method for meta-scheduling and present the architecture of the proposed solution, the existing prediction methods, the neural network prediction algorithm, and the scheduling algorithm. We present the experimental results obtained with our system in the fifth section. In section six we state the conclusions and emphasize directions for future work.

2. RELATED WORK

Grid scheduling is one of the active topics in the literature nowadays. Most of the existing scheduling solutions do not consider the prediction of future behavior of the Grid resources, even if this approach can lead to far better results in optimizing the scheduling decisions. To make the best use of the resources in a shared LSDS environment, an application scheduler should make a prediction of available performance on each resource. Some solutions to develop algorithms based on prediction methods are proposed in (3)(4)(6)(8)(9).

In (3) the authors propose a conservative scheduling technique based on a predictor component to help in making scheduling. The component is able to adjust the scheduling decisions based on a feedback mechanism. The authors compare the prediction accuracy of their method against the ones obtained by using another prediction instrument, Network Weather Service (NWS).

In (4) the architecture proposed uses Globus (5) as an underlying middleware to provide scheduling based on performance predictions. One weakness of both approaches is that, for the most part, they don't consider the local characteristics of individual cluster sites and allow the selection of jobs that can lead to the increase of job queuing time.

The Real-time Scheduling Advisor (RTSA) (6) is a user-level system. An application running on a typical shared, unreserved distributed computing environment can ask advice on how to schedule its compute-bound soft real-time tasks. Given a list of hosts, a description of the CPU requirements of the task, the deadline, and a confidence level, the RTSA recommends one of the hosts and predicts the running time of the task on that host. The solution is based on statistical predictors. In the end, the system presents to the user the confidence intervals for the running time of a task. These confidence intervals are formed using time series analysis of historical information about host load. The disadvantage of the system is, however, that it assumes a homogeneous environment, which is not always true, especially in case of LSDS environments.

An approach similar to the one proposed in this paper is presented in (8). In that paper, the authors are proposing a set of long-term load prediction methods wich reference the properties of processes and the runtime predictions. These methods are used by a prediction module selector that is able to select an appropriate prediction method, based on the actual situation and according to a state of dynamically changing CPU load. Like the approach presented in this paper, their solution is also based on the use of a neural network. However, the scheduling decisions are strictly based on CPU load predictions. Our proposed solutions makes use of a wider range of performance parameters.

In (9) the authors propose the use of a batch-mode scheduling algorithm called First-order Prediction-based Dynamic FPLTF (FP Dynamic FPLTF), abbreviated as FP. It represents a prediction-based scheduling algorithm that works similar to the FPLTF (10) except that it needs the host's latest two load records. The scheduler uses these two records to reconstruct an approximated hosts loading model to predict the host's speed in the future, based on a linear function. FP is able to achieve good performances but, it needs detailed information about tasks and hosts, which is not offered by a Computational Grid composed of a large number of heterogeneous resources.

In fact, the resources in LSDS environments are heterogeneous and widely distributed. This is a problem for many of the previous scheduling attempts to make decisions based on prediction techniques. Previous attempts to develop scheduling solutions based on predictions faced the difficulty of installing monitors to record the performance of the LSDS components and feed a predictor component, a problem that we overcame by using one of the widely used monitoring frameworks today, MonALISA. Another problem faced by previous schedulers is the difficulty in accurately predicting the hosts' characteristics (such as CPU computational power, CPU load, etc). Some of them have proposed techniques to predict the values regarding hosts and tasks by using the Network Weather Service (NWS) (11), a distributed system that periodically monitors and dynamically forecasts the performance of various network and computational resources. The difficulty with this approach comes from the need to install an extra component, in this case the NWS system, to monitor all resources of a LSDS. Our solution uses a monitoring system that is already part of the LSDS infrastructure. The NWS operates a set of forecasting methods that it can invoke dynamically, passing as parameters the CPU load measurements it has taken from each resource. Conversely, the availability prediction is based on a nonparametric method called Binomial Method (12). This method is able to make future availability prediction (with provable confidence bounds) with as few as 20 measurements of the previous availability values. We argue that our solution that is based on neural networks provides better results that the Binomial Method, as presented in the next sections.

3. PREDICTION BASED METHOD FOR META-SCHEDULING IN LSDS ENVIRONMENTS

3.1 Prediction Architecture

In order to achieve complex prediction of various monitored resource parameters or large-scale network events and characteristics, cooperation of many, and possibly heterogeneous, monitoring data collectors distributed over a wide-area network must be accomplished. In such an environment, the processing and correlation of the data gathered at each collector gives a broader perspective of the state of the monitored resources, in which related events become easier to identify.

Therefore, within the MonALISA framework (7), we developed a distributed prediction architecture, able to collect relevant information, present global views from the dynamic set of services running in the distributed environment to higher level services and to further make accurate predictions of the monitored resources' behaviour. Our architectural model consists of Sensors - MonALISA Services executing monitoring tasks, which are able to interact autonomously with other services through dynamic Proxies or via Agents that use self-describing protocols; Web Clients - MonALISA Repositories can dynamically discover these Services by their attributes, interact with them and use the monitored data for further processing, representation of the predicted parameters and automated decision control based on predictions; Web Service Client - uses the WSDL/SOAP interface published by the repository to acces information received from several monitoring Services which will be used for further predictions; Preditction Tool - based on the information obtained through the web services and using the methods presented in the following section develops the parameters predictions; ApMon - is used for injecting the predicted information back into the distributed monitoring system. Figure 1 presents these main components of the architectural model and their communication infrastructure.

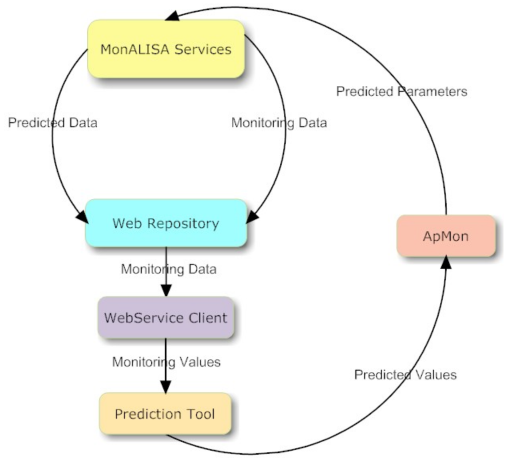

Fig. 1. Prediction Architecture

We detail in the following the main characteristics of the components of our prediction architecture model.

The **MonALISA Service** monitors and tracks site computing farms and network links, routers, and it dynamically loads modules that make it capable of interfacing existing monitoring applications and tools (e.g. Ganglia, MRTG, LSF, PBS, Hawkeye.). The core of the monitoring service is based on a multi-threaded system used to perform the many data collection tasks in parallel, independently. The modules used for collecting different sets of information, or interfacing with other monitoring tools, are dynamically loaded and executed in independent threads. A Monitoring Module is a dynamic loadable unit which executes a procedure (or runs a script / program or performs SNMP request) to collect a set of parameters (monitored values) by properly parsing the output of the procedure. In general a monitoring module is a simple class, which is using a certain procedure to obtain a set of parameters and report them in a simple, standard format. Monitoring Modules can be used for pulling data and in this case it is necessary to execute them with a predefined frequency (i.e. a pull module which queries a web-service) or to "install" (has to run only once) pushing scripts (programs) which are sending the monitoring results (via SNMP, UDP or TCP/IP) periodically back to the Monitoring Service. Allowing to dynamically load these modules from a (few) centralized sites when they are needed makes much easier to keep large monitoring systems updated and to provide new functionalities dynamically; users can also implement easily any new dedicated modules and use it in the MonALISA framework. This architectural model of the Service makes it relatively easy to monitor a large number of heterogeneous nodes with different response times, and at the same time to handle monitored units which are down or not responding, without affecting the other measurements. We use the Service to collect key parameters (ex: Load5, FreeMemory, DiskSpace) which are further used for accurate predictions.

The **MonALISA Repositories** are special types of clients used for long periods storage and further processing of monitoring data. They subscribe to a set of parameters or filter agents to receive selected information from all the Services. This offers the possibility to present global views from the dynamic set of services running in the distributed network environment to higher level services. The received values are further stored locally into a relational database, optimized for space and time. The collected monitoring information is further used to present a synthetic view of how the global system performs. The system targets developing the required higher level services and components of the network management system that provide decision support, and eventually some degree of automated decisions and control.

We developed a special Prediction Repository in our architectural model. We used it to store for long periods of time both the actual monitoring data on which we base our predictions and the actual predictions, for further analysis, fine tuning and eventually some degree of automated decisions and control to higher level services. The Prediction Repository is capable to dynamically plot the parameters we focused on (load, free memory, free disk space, job resources etc.) into a large variety of graphical charts, statistics tables, and interactive map views, following the configuration files describing the needed views, and thus offering customized global or specific perspectives. Within the Prediction Repository we created several views for comparing our prediction models and also different charts to test the accuracy of our predictions showing both the predicted information and the real data. Within the Prediction Repository we developed an automated management systems that uses the predictions. Hence we created special types of data filters, called Actions. They can register for the data produced by the monitoring modules and also for the predictions produces by the Prediction tool, and can take specific actions when some configurable condition is met. This way, when a given threshold is reached, an alert e-mail can be sent, or a program can be run, or an instant message can be issued. Actions represent the first step toward the automation of the decisions that can be taken based on the monitoring and predicted information. It is important to note that Actions can be used in two key points: locally, close to the data source (where monitored data / predictions are produces) where simple actions can be taken like restarting a dead service and globally, in the Prediction Repository, where the logic for triggering the action can be more complicated, as it can depend on several flows of data. This automated management framework becomes a key component. Apart from monitoring and predicting the state of the various LSDS components and alerting the appropriate people of any problems that occur during the operation, this framework can also be used to automate the processes.

We enhanced the Prediction Repository System with fault tolerance capabilities in order to achieve high availability. Hence, we used replication of the Repository Service aiming for a warm standby configuration: in case one repository fails, one or more replicas are ready to take over clients' queries in a transparent way. In this respect we deployed three instances of the repository, located at distinct locations so that local network failures wouldn't affect the system. Deployed repository replicas are permanently aware of each other's current state and after recovery from failure an instance synchronizes its state with the other running replicas ensuring consistency of monitored data.

The **Web Service Client** is used to access data from the Prediction Repository via its published Web Services interface. We have implemented a Web Service client that periodically requests monitoring data from the repository. The client selects the parameters depending on the farm, cluster, node and the parameter name. The number of values that the user is interested in is specified with two time parameters, which describe the amount of time between the return results. Each returned values will have a time-stamp associated with it, which will be later used to compute the predicted value time-stamp.

The time delays values that appear in the communication between framework entities is not significant in the scheduling process because we proposed a method for optimizing advance reservation, so this not have real-time requirements and constrains.

The **Prediction Tool** is using the Web Service client periodically in order to compute the future values for the monitoring parameters. A different client will be used for each parameter for which the user wants to predict the future behavior. The prediction program processes the monitoring parameters values returned by the Web Service client and the computed predictions are sent to the MonALISA service employing the ApMon library, as a new parameter with the appropriate time-stamp. Thus we have developed Java programs that run in a background thread and calculate at regular intervals one-step ahead predictions for the most important monitoring parameters.

ApMon is a set of flexible APIs that can be used by any application to send monitoring information to MonALISA services. The monitoring data is sent as UDP datagrams to one or more hosts running MonALISA services. Applications can periodically report any type of information the user wants to collect, monitor or use in the MonALISA framework to trigger alarms or activate decision agents. We use this tool to inject our predicted information back into the system for further analysis and comparison.

3.2 Prediction methods

There are two types of prediction algorithms that have been developed. Some simple one step ahead prediction methods will be described, together with a neural network based algorithm capable of multi-step ahead prediction. Real data in the form of one of the parameters used to describe the machine load, "Load10", is employed to illustrate the effectiveness of the various prediction algorithms. The time series interval for the monitoring parameters is 1 minute. The prediction algorithms use the last five values to predict the next value (12).

Simple Moving Average. A simple prior moving average (SMA) is the unweighted mean of the previous n values. For example, a 5-minute simple moving average of the Load5 parameter is the mean of the previous 5 minutes values. Considering the values are L_t, L_{t-1}, ..., L_{t-4}, the formula is $SMA = \frac{L_t+L_{t-1}+L_{t-2}+L_{t-3}+L_{t-4}}{5}$. The simple moving average prediction algorithm computes one step ahead point as the mean of the last five values.

Restricted Moving Average. Experimental results have shown that the behavior of the simple moving average prediction algorithm is not the desired one. The predicted value is sometimes an increasing value, and the real value is actually decreasing. To eliminate this behavior, a restriction on the moving average algorithm has been introduced. If the predicted value is higher than the last value, the predicted value will become the last real value that has been provided.

Weighted Moving Average. A weighted moving average (WMA) is an average that has multiplying factors to give different weights to different data points. The weights are decreasing arithmetically as the values are older in time. In an n-value weighted moving average, the last value has weight n, the value before the last has weight $n-1$, and so on: $WMA = \frac{5L_t+4L_{t-1}+3L_{t-2}+2L_{t-3}+L_{t-4}}{5+4+3+2+1}$.

Exponential Moving Average. In an exponential (weighted) moving average (EMA) algorithm, the applied weights are decreasing exponentially. In this way, a more recent observation is given much more importance than the weighted moving average algorithm. In the same time the algorithms does not discard older observations entirely.

The constant smoothing factor α is the degree of weighing decrease and is a number between 0 and 1. If the value at the time t is Lt and the vales assigned to EMA at the same time is S_t, then S_1 will be undefined and S_2 will be initialized as the average of the first 5 values. The formula for calculating the exponential moving average at any time periods $t \geq 2$ is $S_t = \alpha \times L_{t-1} + (1-\alpha) \times S_{t-1}$. Depending on the constant α, older values have more or less importance in the sum. If the smoothing factor α is higher, the older observation are discounted faster.

Random Prediction. The random predictions are based on the standard deviation theory. In probability and statistics, the standard deviation of a set of values is a measure of the spread of its values. In this case, the predicted values will be a random real number in the same interval. The standard deviation (σ) is defined as the square root of the variance. The variance is the average of the squared differences between data points and the mean. Standard deviation, being the square root of units squared, therefore measures the spread of data about the mean, measured in the same units as the data. In other words, the standard deviation is the root mean square deviation of values from their arithmetic mean. The standard deviation for a a sample of the population is calculated using the following formula: $\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$ where N is the number of values, $x_1, x_2, ..., x_N$ are the set of values and is the mean.

The random prediction algorithm computes one step ahead value as a random real number contained in the interval $(\mu - \sigma, \mu + \sigma)$, where μ is the mean of the values.

3.3 Neural Network Prediction Algorithm

Neural networks have been used in economics and financial prediction because of their ability to learn from historical data. Another reason for this choice is that a neural network is also capable of a multi-step ahead prediction model (13). A prediction algorithm using a neural network has been developed as a form of regression. The network will receive as input historical monitoring data and will predict one or more values ahead.

Neural networks are complex modeling techniques that are able to shape nonlinear functions. They are used in situations when there is a dependency between the known input and the unknown output, but the nature of that relation cannot be expressed exactly. In the case of CPU Load5 prediction, the future value will always depend on the current and the past behavior, but is impossible to define the exact relation between those two. This makes the use of neural network ideal for a prediction tool (14)(15).

One Step-Ahead Prediction. One step-ahead prediction is produced by a neural network with one neuron in the output layer. The network is trained to predict the next value of a monitoring parameter, such as CPU Load5 or Free Memory, based on its last five values. The model can be extended for any monitoring information, the accuracy of the predictions depending on the data's behavior in time.

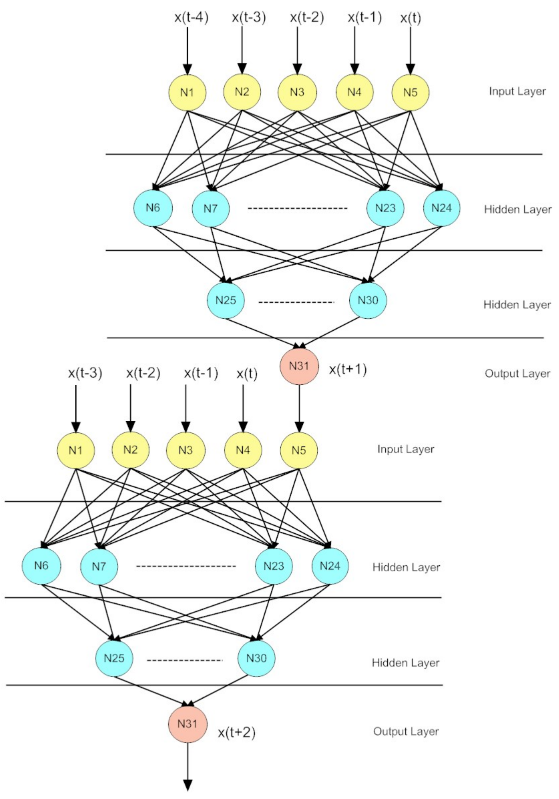

Fig. 2. Two step ahead prediction with a neural network

The Neural Network's Structure. The structure of the network has been established based on a trial and error approach. The number of input neurons has been chosen as a consequence of the simple moving average prediction. Real-time tests have shown that five values are sufficient for evaluating the performance of one parameter for a short period of time. The number of hidden layers as well as the number of neurons in each of the hidden layer was established by conducting tests. Two hidden layers can approximate any non-linear function that would describe the relationship between past and current values and future behavior. The numbers of neurons in the hidden layers are 19, and 5 respectively.

Pre- and Post- Processing. Although all the monitoring parameters are numeric and thus appropriate to be passed to the network, some pre-processing and post-processing must be done. Due to the sigmoid activation function used, the output is in the range (0, 1). As a consequence, some scaling algorithm must be applied to ensure that the network's output will be in the right range. The simplest scaling function is minimax: this finds the minimum and maximum values of a variable in all training data, and performs a linear transformation (using a shift and a scale factor) to convert the values into the target range (typically [0, 1]). Experiments have shown that because of the wide range of the variables, the resulted predictions have a low precision. In order to improve the predictions, a different scaling approach has been implemented. Instead of finding the minimum and maximum values in all training data, the patterns are considered individually. The minimum and maximum values required for the scaling algorithm are computed as $\mu - \sigma$ and $\mu + \sigma$ respectively, where μ is the mean of the set of values in one pattern and σ is the standard deviation. Using this scaling algorithm, the accuracy of the prediction is significantly better then when using the minmax function.

Training the Network. The neural network is trained using the back propagation algorithm. The patterns in the training set are selected from the historical monitoring data. If the interval between values is one minute, then a training set must contain at approximately two hours of gathered monitoring information.

The historical data is provided by the Web Service client. The training patterns are chosen at run time, given that the performance of the resources can change over time. The large amount of data is divided into input and target patterns and passed to the neural network for training. The back propagation algorithm will analyze the training set a large number of times, until the network's error will reach an acceptable level. After the neural network is trained, recent historical data is passed to the network and it produces the next value of that particular parameter. In the case of the program that runs as a background thread, the process is repeated every minute in order to predict periodically the next future value.

Multiple Step Ahead Prediction. In the case of multiple step ahead prediction, the same neural network is used. After the next value in a time series is produced, the following values can be estimated by feeding the newly generated value back into the network with other values.

Figure 2 shows an example of a two step ahead prediction and how is generated. In order to predict two steps ahead, the procedure needs the last five values. The prediction process begins at the current moment t, and is trying to predict the value at the time $t + 1$. The predicted value is considered correct and it is fed back into the network as an input, along with the shifted past values. The process is repeated and the network produces the predicted value for the time $t + 2$.

3.4 Prediction Method for Resource Reservation

After negotiation phase a Service Level Agreement (SLA) is established between the Broker and the Agent that represents the resources on which the job will be executed. Through this SLA, the Broker agrees to pay the price asked by the Agent for the execution of the job and the Agent agrees to deliver the results obtained by executing the job in the specified time. Any violation of the SLA by the Agent in an Economic Grid conducts to a bad rating from the Broker, eliminating it from the selection process. To ensure that the SLA is respected by the Agent the resource reservation mechanism needs to be implemented using an accurate performance prediction of the resources when the job will be actually executed.

The reservation method used by the Agent can be improved by predicting the running time of the jobs on resources with different level of performance at a given time. The prediction will be based on the instance based learning technique. This technique will use a training database (for each executed job, the attributes - user name, number of CPUs needed to execute the job, the input parameters, etc, and the execution time will be saved in the database) to find the similar jobs with the query point. Once the set of nearest neighbors are identified the runtime of the q job will be predicted using the weighted average of the nearest n neighbors:

$$P(q) = \frac{\sum_{i=1}^{n} W_i Val(T_i)}{\sum_{i=1}^{n} W_i}$$

where e_i it the i^{th} nearest neighbor and $Val(T_i)$ is the recorded runtime for T_i job. We consider $K(d)$ the function that calculates a weight from the distance and k is a smoothing factor:

$$W_i = K(D(q, T_i))$$

$$K(d) = \exp -(\frac{d}{k})^2$$

The relevance between two jobs with input attributes vector x and y is measured by the distance function:

$$D(x, y) = \sqrt{\frac{\sum_{i=1}^{m} w_i d_i(x_i, y_i)}{\sum_{i=1}^{m} w_i}}$$

where w_i is the weight of attribute i, and the d_i is: *overlap* if a is a nominal attribute, *nsdiff* if a is a numeric scalar or *nvdiff* if a is a numeric vector. The overlap and *nsdiff* are used to determine job similarity using:

$$nsdiff_i(x, y) = \frac{|x - y|}{max_i - min_i}$$

$$overlap(x, y) = \delta_{xy}$$

where max_i and min_i are the maximum and the minimum observed values for attribute i. The resource state similarity is determined using the following formula:

$$nvdiff_i(x, y) = \frac{\sum_{a=1}^{N_i} |x_a - y_a|}{range(i)}$$

where a is the a^{th} category, the N_i is the total number of categories of x and y, $range(i)$ is the maximum value of difference in the training data for attribute i. The recorded resource status for the executed jobs will be categorized by group name, user name and queue name to minimize the search in the training data set. The history size, neighbor size and smoothing factor are determined by trial-and-error, to minimize the average prediction error.

3.5 Prediction Based Meta-Scheduling Algorithm

In the case of a static scheduling approach, resources are allocated as soon as the request arrives at the Agent. When dealing with task dependencies, it is possible that some tasks will have the start time in the future, due to precedence constrains. In the case of DAG scheduling, the nodes and edges are obtained by estimation at compile time. Based on the requirements of each tasks and available resources the start time for each task can be estimated.

We present in section 2 the related work describing the latest tentative to elaborate a scheduling algorithm based on predictions. We propose a method to use the prediction component described in the last section in a distributed

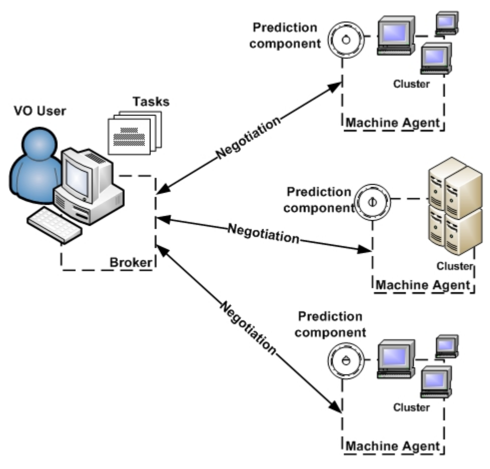

Fig. 3. Scheduling Framework

framework. The proposed prediction-based meta-scheduler uses the two entities model for the scheduler architecture (see figure 3).

When a user wants to execute a job, the job requirements are sent to the Broker. The Broker will then start a communication with all the available Agents (computational nodes) in the system. Because the application is an economic one, the Brokers and Agents will negotiate the price for the needed/offered resources. When the Agent receives a request from a Broker, it will evaluate the jobs requirements to see if it can execute the job in the required time. If it can execute the job in the required time, the Agents send back to the Brokers the offers that contain the estimated price for using their resources. The Brokers receive the offers from Agents, and choose the resources with the best offer for the user.

The Broker collects the users requests and the start time for each task is determined. In the case when the start time is in the future, rather than at the current moment, the Agent uses the predicted state of the system. Using this approach, an optimal scheduling solution can be achieved in a static manner. This way the much more expensive dynamic scheduling approach can be avoided. When the start time for each task is known, the resource status predictions can be used to schedule the task in a static approach. The Agent running the scheduler algorithm can use the estimated resource status to match them with the submitted tasks.

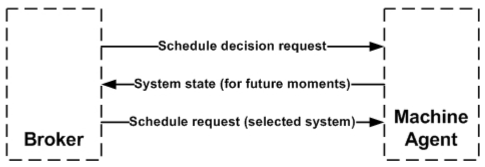
Schedule decision request
System state (for future moments)
Schedule request (selected system)
Broker
Machine Agent

Fig. 4. Communication between framework entities

Here is a description of the proposed scheduling algorithm in a logical flow of activities:

Step 1. A user requests that one or more tasks are scheduled. His request has as a parameter the name of the file containing a description of the tasks. The file has a standard XML format and presents requirements for each task relative to memory, cpu usage, execution time, etc. An important parameter specifies the start time and another specifies the completion time for each task.

Step 2. The input file is processed and a "batch of tasks" (group of tasks) object is constructed.

Step 3. The batch of tasks is broadcast to all the nodes in the cluster (Machine Agents).

Step 4. The nodes receive the group of tasks to be scheduled. Each Machine Agent analyzes the task description and predicts the system state at the start time moment, indicated in task's description.

Step 5. Each Node send to Broker the analysis results. The response is represented by a set of tasks identification that could be executed on that machine.

Step 7. The Broker make the system selection for each task.

Step 6. The scheduling obtained is saved in a history file on each node in the cluster.

The next section presents the experimental results of testing the prediction methods. The very good results for prediction methods sustains the correctness for the Step 4 from the described algorithm.

4. EXPERIMENTAL RESULTS

In order to evaluate the accuracy of the prediction methods, a Java class has been implemented for each algorithm. The program runs as a background thread providing real time prediction for some of the most important monitoring parameters.

Most of the parameters have a very instable behavior. Consequently, the predictions are rarely 100 percent accurate, but rather give a good estimate of near future behavior. In this sense, various parameters were chosen in order to test the prediction algorithm. The test benches includes parameters with high variation in time and parameters with a periodical behavior.

4.1 Prediction results

We will present the results of the experimental tests made for "Load10" parameter. While it is easy to implement and requires no additional overhead, its performance is marginally satisfactory. By the nature of the calculation, this algorithm produces results that are both delayed and dampened. The algorithm has a tendency to flatten local peaks as a result of the averaging function, but the result generally follows the real trends.

Simple Moving Average. While it is easy to implement and requires no additional overhead, its performance is marginally satisfactory. By the nature of the calculation, this algorithm produces results that are both delayed and dampened. The algorithm has a tendency to flatten local peaks as a result of the averaging function, but the result generally follows the real trends (see figure 5).

Restricted Moving Average. When using this algorithm, the peak sensitivity problems persist and the restriction itself is a big source of errors, especially in the case of fast, high amplitude variations (see figure 6). In some cases the predicted values and the real ones show opposite trends (e.g. the real value increases but the predicted value is lower than the previous one).

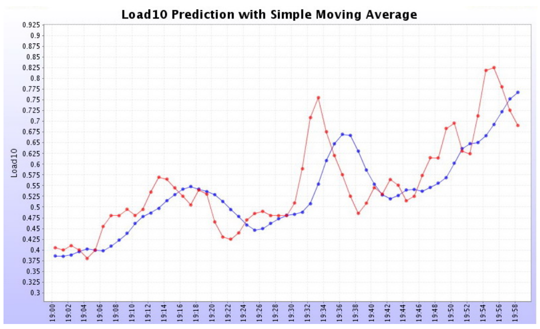

Fig. 5. Load10 real (red) and predicted (blue) values using a simple moving average

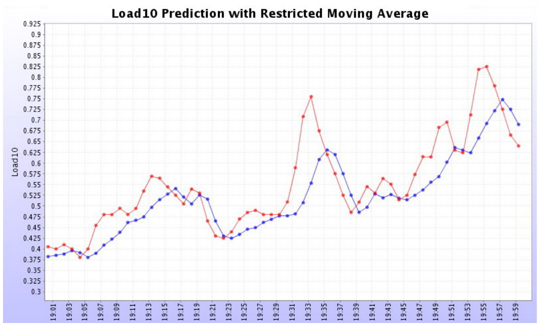

Fig. 6. Load10 real (red) and predicted (blue) values using a restricted moving average

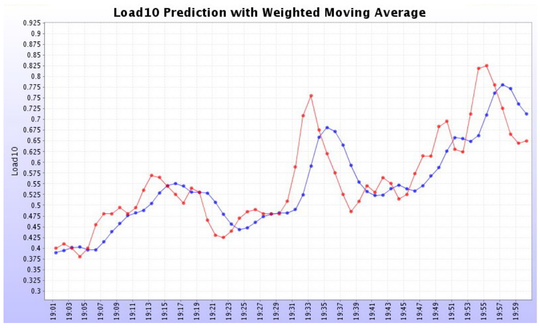

Fig. 7. Load10 real (red) and predicted (blue) values using a weighted moving average

Weighted Moving Average. Although this prediction algorithm is giving extra weight to more recent data points, the predicted values are not very accurate. The reason is that the recent past may not offer sufficient information with regard to the next value of the monitored parameter. The weighting procedure assumes that more recent data is more significant, which in some cases may not be the case, for example for rapid fluctuations (see figure 7).

Exponential Moving Average. While the performance is generally better than the one expected from a simple moving average algorithm, this method fails to produce accurate results when there is a significant difference between values at consecutive time points. Good results can be achieved by tuning the smoothing factor α, if the general behavior of the signal is known. For a completely random signal, like Load10, the results are only slightly better than the ones produced by the moving average technique, with the greatest error being produced mostly

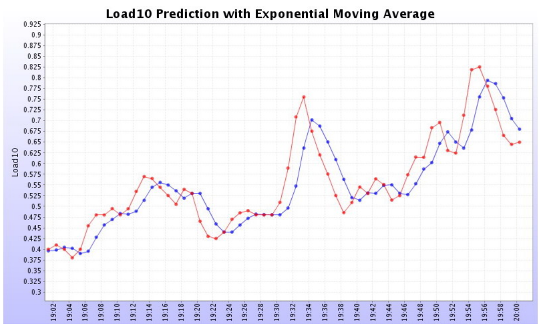

Fig. 8. Load10 real (red) and predicted (blue) values using a exponential moving average with the constant $\alpha = 0.6$

when the signal varies abruptly after a period of little or no change (see figure 8).

Random Prediction. Intuitively, it can be seen that this method is well suited for predicting signals with small variations. However, the real Load5 parameter has a variation that is seldom small, meaning that the performance of this algorithm can decrease dramatically when the signal changes rapidly (see figure 9).

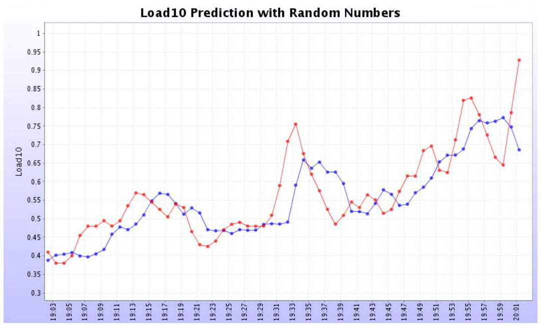

Fig. 9. Load10 real (red) and predicted (blue) values using a random prediction

Neural Network Prediction. Neural networks have been used with success in problems concerning time series prediction. They process records one at a time, and ”learn” by comparing their prediction of the record (which, at the outset, is largely arbitrary) with the known actual record. The errors from the initial prediction of the first record is fed back into the network, and used to modify the networks algorithm the second time around, and so on for many iterations. An artificial neural network provides an efficient technique for a multi-step ahead prediction. If large amount of historical data is available, a neural network is capable of discovering hidden dependencies between values at fixed time intervals without the need of other information.

Load10 parameter, that shows the load of the system in the last 10 minutes, generally has large variations and its behavior is rarely repetitive, therefore the training set cannot include all the situations. According with this, the prediction achieves good performance in the case of large variations of Load10 parameter (see figure 10).

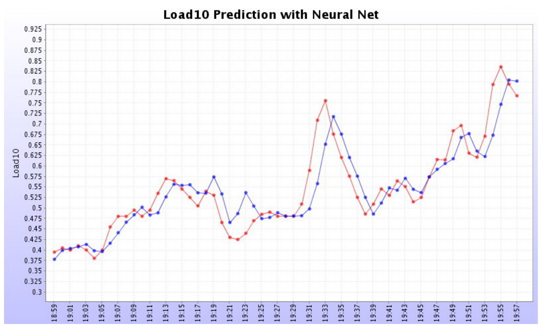

Fig. 10. Load10 real (red) and predicted (blue) with neural network algorithm

4.2 Errors Interpretation

This section discusses the experimental results obtained by our new algorithm in the situation of one-step ahead prediction for the earlier described parameters. The results are also compared with those obtained by other algorithms on the same historical data set, comparison made using the absolute percentage error criterion. The presented graphics highlight the errors obtained by the following prediction approaches:

- SMA (simple moving average) represented with purple
- EMA (exponential moving average) represented with blue
- WMA (weighted moving average) represented with green
- Rand (random prediction) drawn using dark blue
- Cascor (the prediction algorithm proposed in (2) based on the Cascade Correlation NN and genetic algorithms) drawn with dark green
- FA (the new prediction approach based on decomposition using a perceptron) represented using red
- FC (the new prediction approach based on decomposition using the Cascade Correlation NN architecture) represented using orange

Table I presents the absolute percentage errors obtained using those prediction algorithms for different system parameters where the minimum error for each parameter is bolded. As the table emphasized, in each situation the minimum error is obtained using the new prediction approach using the Cascade Correlation NN architecture. This table presents a comparison with other optimization techniques based on prediction, applied in large scale distributed systems.

Figure 11 presents a comparison between the errors obtained for the 'idle' parameter using different prediction algorithms. Because of the fact that the parameter's behavior don't vary very fast, the errors obtained are very small (less than 0.12% for all algorithms). There aren't large differences between the results obtained using the new prediction approach using a perceptron (marked with FA in the graphic) and the results obtained by the new algorithm using a Cascade Correlation NN architecture (marked with FC in the graphic). The difference between those two is 1.8%. The improvement offered by the new approach in comparison with the algorithm proposed in (2) is more than 51% and more than 90% in comparison with the classical prediction algorithms. This scenarios is important for scheduling oprimization because highlights that the tasks' idle time is reduced significantly by using the predictions.

We can conclude that the performance of last mentioned methods is better than the firsts methods. Neural networks are selected for prediction because they are capable of a multi-step ahead prediction with minimum error value. Even they are slower than other prediction methods (e.g. regression), having a higher computation times than regression, the values for errors make this method to be a better one.

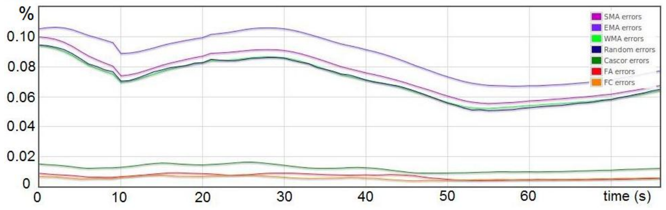

Fig. 11. Errors' evolution for the 'idle' parameter

5. CONCLUSIONS

The present paper describes several prediction methods for use in LSDS and distributed system environments. Moreover, an original contribution to prediction methods, which uses a neural network has been developed, and its structure and functionality are explained. The project considers time series prediction of the status of the distributed system resources. Historical monitoring data provided by a monitoring service is used to predict one or more values into the future. The predictions are used in a meta-scheduler algorithm, improving this way the resource utilization balance. LSDS computing has become an active research area due to the increasing number of scientific applications that are both computing and data intensive. We also present the architecture developed for the prediction system in this project, along with the constituent modules. The predictions are based on historical monitoring data provided by MonALISA monitoring system.

Simple prediction algorithms have been implemented and analyzed as a part of this project. The algorithms are based on classical statistical methods such as moving average and standard deviation. Some of the algorithms predict the next values as the moving average of the last five values. We used simple moving average as a prediction method, along with a restricted version of the same algorithm, weighted and exponential moving average. Random predictions based on the mean and the standard deviation are also evaluated. The resulted values were compared in time with the real measured value in order to investigate the predictions accuracy.

Neural networks have been used with success in problems concerning time series prediction. An artificial neural network provides an efficient technique for a multi-step ahead prediction. The behavior of resources, although sometimes periodical, is always non-linear, which makes the classic linear regression methods obsolete. If large amount of historical data is available, a neural network is capable of discovering hidden dependencies between values at fixed time

Table 1. Average Percentage Errors (%) for One-Step Ahead Prediction

Parameter	SMA	EMA	WMA	RAND	CASCOR	FA	FC
idle	0.10	0.12	0.10	0.10	0.02	0.01	**0.01**
cpu_usage	6.67	7.28	6.90	8.47	6.43	6.99	**5.65**
load5	0.72	0.83	0.56	0.91	0.48	0.46	**0.27**
free mem	12.11	12.77	9.89	14.27	12.46	8.75	**7.40**

intervals without the need of other information. Because of these reasons, we have also implemented a prediction system based on a neural network. The neural network predictions have been analyzed in comparison with the simple prediction algorithms. Predicting the future status of the resources composing a distributed system is important in a highly dynamic system. An optimum task scheduling based on resources state prediction will have an immediate effect on the overall balance.

The selection of the system that will receive a submitted job for processing is based on the detailed information about the available resources to match the job's requirements. The prediction based meta-scheduler can significantly improve the time necessary to schedule tasks with dependencies. Using the predictions, we have been able to state that the idle time of tasks with the start time in the future can be reduced significantly. Reducing the idle time would result in resource utilization and load balance that are close to the optimum.

Future work will consider the optimization for multi-steps prediction method and the possibility of tunning the scheduling algorithm.

REFERENCES

[1] Florin Pop, Alexandru Costan, Ciprian-Mihai Dobre, Valentin Cristea. 2009. Prediction Based Meta-Scheduling for Grid Environments, the 17th International Conference on Control Systems and Computer Science, pp 330-337, Bucharest, Romania, May 2009.

[2] Andreea Visan, Mihai Istin, Florin Pop, and Valentin Cristea. 2010. Automatic Control of Distributed Systems Based on State Prediction Methods. In Proc. of the 2010 Int. Conference on Complex, Intelligent and Software Intensive Systems (CISIS '10). IEEE Computer Society, Washington, DC, USA, 502-507.

[3] Lingyun Yang, Jennifer M. Schopf, and Ian Foster. 2003. Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments. In Proceedings of the 2003 ACM/IEEE conf. on Supercomputing (SC '03). ACM, New York 31-.

[4] Junwei Cao, Stephen A. Jarvis, Daniel P. Spooner, James D. Turner, Darren J. Kerbyson, and Graham R. Nudd. 2002. Performance Prediction Technology for Agent-Based Resource Management in Grid Environments. In Proc. of the 16th Int. Parallel and Distributed Processing Symposium (IPDPS '02). IEEE Computer Society, Washington, DC, USA, 265-.

[5] Ian Foster. 2005. "Globus Toolkit Version 4: Software for Service-Oriented Systems", IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp. 2-13, 2005.

[6] Peter A. Dinda. 2002. A Prediction-Based Real-Time Scheduling Advisor. In Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS '02). IEEE Computer Society, Washington, DC, USA, 35-.

[7] Iosif Legrand, Ramiro Voicu, Catalin Cirstoiu, Costin Grigoras, Latchezar Betev, and Alexandru Costan. 2009. Monitoring and Control of Large Systems with MonALISA. Queue 7, 6, Pages 40 (July 2009).

[8] Y.Sugaya, H. Tatsumi, M. Kobayashi, H. Aso. 2008. Long-Term CPU Load Prediction System for Scheduling of Distributed Processes and its Implementation. In Proceedings of the 22nd International Conference on Advanced Information Networking and Applications (AINA '08). IEEE Computer Society, Washington, DC, USA, 971-977.

[9] Elisa Heymann, Miquel A. Senar, Emilio Luque, and Miron Livny. 2000. Adaptive Scheduling for Master-Worker Applications on the Computational Grid. In Proc. of the First IEEE/ACM Int. Workshop on Grid Computing (GRID '00), Rajkumar Buyya and Mark Baker (Eds.). Springer-Verlag, London, UK, 214-227.

[10] Fabricio A. B. da Silva and Hermes Senger. 2011. Scalability limits of Bag-of-Tasks applications running on hierarchical platforms. J. Parallel Distrib. Comput. 71, 6 (June 2011), 788-801.

[11] Rich Wolski, Neil T. Spring, and Jim Hayes. 1999. The network weather service: a distributed resource performance forecasting service for metacomputing. Future Gener. Comput. Syst. 15, 5-6 (1999), 757-768.

[12] J. Brevik, D. Nurmi, and R. Wolski. 2004. Automatic methods for predicting machine availability in desktop Grid and peer-to-peer systems. In Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid (CCGRID '04). IEEE Computer Society, Washington, DC, USA, 190-199.

[13] F. Virili, B. Freislenben. 2000. Nonstationarity and Data Preprocessing for Neural Network Predictions of an Economic Time Series. In Proceedings of the IEEE-INNS-ENNS Int. Joint Conf. on Neural Networks (IJCNN'00)-Volume 5 - Volume 5 (IJCNN '00), Vol. 5. IEEE Computer Society, Washington, DC, USA, 5129-.

[14] Jovina Roman and Akhtar Jameel. 1996. Backpropagation and Recurrent Neural Networks in Financial Analysis of Multiple Stock Market Returns. In Proceedings of the 29th Hawaii International Conference on System Sciences Volume 2: Decision Support and Knowledge-Based Systems (HICSS '96). IEEE Computer Society, Washington, DC, USA, 454-.

[15] Danilo P. Mandic and Jonathon Chambers. 2001. Recurrent Neural Networks for Prediction: Learning Algorithms,Architectures and Stability. John Wiley & Sons, Inc., New York, NY, USA.