J Supercomput (2015) 71:824–839
DOI 10.1007/s11227-014-1327-2

Resource and application-aware resource discovery in computing environments

Mohammad Norouzi · Ali Jannesari

Published online: 14 November 2014

Abstract Efficient resource discovery plays a vital role in the effective management of resources and applications in heterogeneous computing environments. Therefore, the knowledge of applications' behavior and resources' usage pattern improves resource discovery decisions. This knowledge can be provided for the resource discovery mechanism by cooperating with the load balancing mechanism. In this paper, we formulate their cooperation by considering some parameters that represent applications' behavior and resources' usage patterns and extract the relation between them to introduce a formula using mathematical methods. Further, the resource discovery mechanism uses the formula to predict resources' load before assigning them new processes and thus it prevents resource overloading which happens frequently in computing environments.

Keywords Computing environments · Resource discovery · Load balancing · Application process behavior · Resource capacity · Multivariate regression

1 Introduction

Regarding today's science and industry interest to reach higher computing power, systems that aim to provide such information processing rate are growingly analyzed,

M. Norouzi (✉)
Iran University of Science and Technology, Tehran, Iran
e-mail: gmnorouzi@gmail.com; noroozi@comp.iust.ac.ir

A. Jannesari
German Research School for Simulation Sciences, Aachen, Germany
e-mail: a.jannesari@grs-sim.de

A. Jannesari
RWTH Aachen University, Aachen, Germany

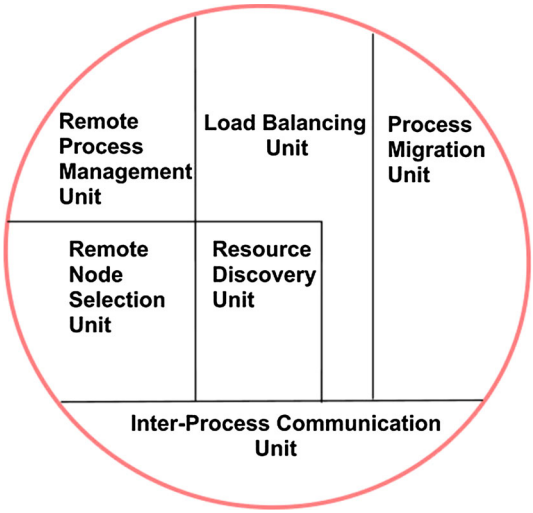

Fig. 1 Some mechanisms in a distributed computing environment

designed and implemented [1]. Examples range from Grid, Cluster, Peer-to-Peer computing environments and to multi- and many-core processors. One of the biggest issues in such environments is efficient discovery of resources for applications' processes, namely resource discovery. There are many issues including heterogeneity, dynamicity and failure of resources that make resource discovery a challenging process. In addition to resource-related issues, different classes of applications and resource usage patterns complicate this challenge. Also, Network-related issues add up to the problem.

On the other hand, there are several mechanisms in a distributed computing environment. Figure 1 shows some of the common mechanisms. Each of the mechanisms is designed for a certain goal and obtains related knowledge about the environment to fulfill its goal. For example, load balancing mechanism balances load on resources and it monitors resources status information, applications' behavior and the environment's policies to maintain this goal. Thus, it has a thorough knowledge about the mentioned aspects of the computing environment. However, this knowledge can improve resource discovery decisions significantly since the mechanism discovers suitable resources for running processes considering applications requirements and resources capacities and usage policies, [2]. In addition, resource discovery mechanism can improve its decisions by collaborating with inter-process communication mechanism. This mechanism has significant knowledge about the network including its topology, the nearness of nodes to each other and network traffic and congestion. Moreover, remote node selection mechanism finds a node which possesses a required resource from outside if no suitable resource is present or available in the environment. Therefore, this mechanism can provide resource discovery mechanism with a good knowledge about the outside environment.

However, other mechanisms can act accordingly and improve their decisions. Process migration mechanism migrates a process to another resource and remote process management mechanism is responsible for installation and configuration of all the mentioned mechanisms in a newly joint node.

Anyhow, we have already researched the impact of the load balancing knowledge on resource discovery decisions in [2]. We proposed considering some parameters that represent the behavior of applications as well as some parameters that characterize resources' usage patterns. In this paper, we are going to extract the relation between those parameters using regression analysis and derive a formula. Indeed, regression is a statistical process for estimating the relationships among variables. There are two types of variables in a regression function namely dependent and independent. More specifically, regression helps one understand how the typical value of the dependent variable varies when any one of the independent variables is changed. Thus, resources load can be predicted before running new processes on them using this formula and their workload is automatically balanced; avoiding resource overloading.

Further, we can consider network and QoS parameters and improve resource discovery decisions accordingly. In addition, each node makes resource discovery decisions locally based on the formula which prevents transmission of resources information. Simulation results showed that the mechanism increased scalability in size by 7 to 15 %, reduced message transmission rate by 15 % and improved hit rate by 51 %, [2].

The rest of the paper is organized as follows. Section 2 discusses related works. Sections 3 and 4 introduce the formula and show the experimental results and finally, Sect. 5 concludes the paper.

2 Related work

Many resource discovery and load balancing mechanisms have been introduced so far. Concerning various computing environments and their design and implementation purposes, these mechanisms are categorized. Resource discovery and load balancing operations are carried out by one single mechanism called the scheduling in Clusters like *PBS*, *Maui*, *Torque*, *Mosix* and*Condor*. These Cluster managers are based on a central structure and, therefore, they suffer from low scalability, [3]. On the other hand, resource discovery and load balancing operations are carried out by two separate mechanisms in Grids and Cluster of clusters environments such as *Mosix2* and *Moab* [4]. In spite of Clusters, scalability is not the main obstacle of the latter environments, but they require collecting and transferring a huge amount of resources information periodically or based on some events to make resource discovery and load balancing decisions, [5].

Being distributed purely, p2p environments are scalable and they do not need to transfer resources' information, [6]. Thus, many p2p-based resource discovery and load balancing mechanisms have been introduced for Cluster and Grid environments, [7,8]. However, these mechanisms neglect the fact that there are other mechanisms in the computing environment that possess significant knowledge about the environment, the node and its resources that can assist them in making more precise decisions. In [2], the authors have introduced a model for communication between resource discovery and load balancing mechanisms in computing environments. They have proposed to extract the behaviors of running processes and make resource discovery decisions according to them in addition to the loads of resources. As they have proposed, the load balancing mechanism is in charge for the extraction of processes' behavior and

Table 1 Applications behavioral parameters according to resource types

Attribute resource	Static	Dynamic
Processor	• Processor_Scheduling_Algorithm	• Process_Required_Load
Memory	• Memory_Type	• Memory_Size_Request
	• Memory_Read_Speed	• Memory_Access_Pattern
	• Memory_Write_Speed	
I/O	• I/O_Device_Type	• I/O_Usage_Pattern
File	• File_System_Type	• File_Access_Rate
	• FileSys_Access_Type	

it can utilize several techniques including simple source code analysis, sequential access pattern extraction, history-based approaches, and on-line behavior extraction [9]. Therefore, they have devised a method to prevent resource overloading which occurs frequently in computing environments, in addition to the other mentioned benefits. However, they have named some parameters that represent the behaviors of processes as well as the resources' load, but they have not pursued to represent a formula that maintains the relation between the parameters so that the formula can be used by resource discovery mechanism.

3 The proposed formula

We have proposed in [2] that 2 types of parameters improve resource discovery decisions including applications behavior and resources capacities and usage patterns, [2]. Table 1 shows some applications' behavioral parameters.

As proposed in [2], load balancing mechanism is responsible to extract the behavior of processes in each node and obtain the actual values of parameters in Table 1. For example, *Processor_Scheduling_Algorithm* is a static attribute of processors and *Process_Required_Load* is a dynamic attribute. To show how this behavior would improve resource discovery decisions consider a batch process. The idle situation for this process is when the discovered processor's scheduling algorithm is also batch, such as FCFS. In addition, *Process_Required_Load* indicates how heavily the process will use the processor. Therefore, resource discovery mechanism should discover a processor which its current processes let the required load happen. Like processor, load balancing mechanism extracts attributes of other resource types and feeds resource discovery mechanism. Table 2 shows some sample values for parameters in Table 1 (Currently Light or Currently Heavy values are set by the environment's administrators).

Once a process requires a resource, load balancing mechanism prepares a request containing these attributes. Then, resource discovery mechanism considers resources that have the mentioned static attributes merely and uses dynamic attributes to predict their load assuming the new process is running on them. However, load prediction is a function of current load on resources and the load generated by the new process. Both these loads can be calculated; the new load is estimated by behavior of the process and current load is obtained by calculating capacities of resources against how much they are being used by their current processes.

Table 2 Applications behavioral parameters and their sample values

Resource type	Behavioral parameter	Sample values
Processor	Process_Required_Load	'Do not Care' | 'Currently Light'| 'Currently Heavy'
	Processor_Scheduling_Algorithm	'Batch' |'Interactive'
Memory	Memory_Type	'Main' |'Secondary'
	Memory_Read_Speed	1 MHz
	Memory_Write_Speed	10 KHz
	Memory_Size_Request	10 MB
	Memory_Access_Pattern	'Currently Heavy' | 'Currently Light'
I/O	I/O_Device_Type	'Network Bandwidth'
	I/O_Usage_Pattern	'Do not Care' | 'Currently Light' | 'Currently Heavy'
File	FileSys_Access_Type	'Sequential' |'Random'
	File_Access_Rate	'Heavy' | 'Light'
	File_System_Type	'ext2' |'ext3' | 'ext4'|'NTFS'

Table 3 Some important parameters that influence resource usage patterns

Attribute Resource	**Static**	**Dynamic**
Processor	• Processor_Type	• Processor_Usage
	• Processor_Speed	•Processor_Queue_Length
Memory	• Main_Memory_Size	• Used_Main_Memory
	• Page_Size	• Used_Swap_Memory
	• Swap_Memory_Size	
	• Method_Used (e.g., paging, segmentation)	
I/O	• I/O_Device_Data_Transmission_Rate	• I/O_Device_Queue Length (e.g., network)
File	• Disk_Management_Algorithm	• Disk_Queue_Length
	• Disk_Size	
	• Swap_Size	

We have categorized them according to their type and dynamicity

Parameters in Table 3 are used to calculate current load on a resource. Again, we can categorize them according to the types of resources and their static or dynamic attributes. Static attributes are used to calculate capacities of resources and dynamic attributes show how much they are being used by current processes.

Now, we can formulate load prediction considering parameters in Tables 1 and 3. The general form of formulae for each resource type is as the following:

$$Decision = e(Processor\_Predicted\_Load, Memory\_Predicted\_Load, I/O\_Predicted\_Load, File\_Predicted\_Load) \quad (1)$$

$$Processor\_Predicted\_Load = f(Processor\_Speed, Processor\_Current\_Load, Processor\_Queue\_length, Process\_Required\_Load) \quad (2)$$

$$Memory\_Predicted\_Load = g(Main\_Memory\_Size, Page\_Size, Swap\_Memory\_Size, Method\_Used, Used\_Main\_Memory, Used\_Swap\_Memory) \tag{3}$$

$$I/O\_Predicted\_Load = h(I/O\_Data\_Transmission\_Rate, I/O\_Queue\_Length) \tag{4}$$

$$File\_Predicted\_Load = i(Disk\_Management\_Algorithm, Disk\_Size, Swap\_Size, Disk\_Queue\_Length) \tag{5}$$

The scope of our proposed resource discovery mechanism is limited to computing environments and thus we only explain how the formula for *Processor_Predicted_Load,* i.e., function f in Eq. (2) is obtained hereafter.

Regression analysis extracts the relationship between dependent and independent variables based on real data samples. To gain the real data, we have conducted an experiment and consequently kept track of changes that have been made in *Processor_Predicted_Load* (as the dependent variable)*, Process_Required_Load*, *Processor_Current_Load*, *Processor_Queue_Length* and *Processor_Speed* (as independent variables). We have used least squares regression method to represent the relation between the parameters. Equation (6) shows the regression function with the variables.

$$\begin{aligned} &Processor\_Predicted\_Load \\ &= \beta_0 + \beta_1 \times Process\_Required\_Load + \beta_2 \times Processor\_Current\_Load \\ &\quad + \beta_3 \times Processor\_Queue\_Length + \beta_4 \times Processor\_Speed \end{aligned} \tag{6}$$

We need to calculate the values of β_0 to β_4 in Eq. (6) to obtain the final format of the formula. In this case, regression analysis proposes an equation set where β_0 to β_4 are considered as variables and the real data that we have obtained in experiments as constants. Equations (7) to (11) constitute the equation set. The series in Eqs. (7) to (11) are replaced with constant values obtained in experiments. In other words, the series are sum of the real data we have obtained during experiments. Therefore, the result is five equations which have β_0 to β_4 as variables.

$$\begin{aligned} \sum_{i=1}^{n} Processor\_Predicted\_Load_i &= \beta_0.n + \beta_1.\sum_{i=1}^{n} Process\_Required\_Load_i \\ &+ \beta_2.\sum_{i=1}^{n} Processor\_Current\_Load_i + \beta_3.\sum_{i=1}^{n} Processor\_Queue\_Length_i \\ &+ \beta_4.\sum_{i=1}^{n} Processor\_Speed_i \end{aligned} \tag{7}$$

$$\sum_{i=1}^{n} Process\_Required\_Load_i \times Processor\_Predicted\_Load_i$$
$$= \beta_0 . \sum_{i=1}^{n} Process\_Required\_Load_i + \beta_1 . \sum_{i=1}^{n} Process\_Required\_Load_i^2$$
$$+ \beta_2 . \sum_{i=1}^{n} Process\_Required\_Load_i \times Processor\_Current\_Load_i$$
$$+ \beta_3 . \sum_{i=1}^{n} Process\_Required\_Load_i \times Processor\_Queue\_Length_i$$
$$+ \beta_4 . \sum_{i=1}^{n} Process\_Required\_Load_i \times Processor\_Speed_i \quad (8)$$

$$\sum_{i=1}^{n} Processor\_Current\_Load_i \times Processor\_Predicted\_Load_i$$
$$= \beta_0 . \sum_{i=1}^{n} Processor\_Current\_Load_i$$
$$+ \beta_1 . \sum_{i=1}^{n} Processor\_Current\_Load_i \times Process\_Required\_Load_i$$
$$+ \beta_2 . \sum_{i=1}^{n} Processor\_Current\_Load_i^2$$
$$+ \beta_3 . \sum_{i=1}^{n} Processor\_Current\_Load_i \times Processor\_Queue\_Length_i$$
$$+ \beta_4 . \sum_{i=1}^{n} Processor\_Current\_Load_i \times Processor\_Speed_i \quad (9)$$

$$\sum_{i=1}^{n} Processor\_Queue\_Length_i \times Processor\_Predicted\_Load_i$$
$$= \beta_0 . \sum_{i=1}^{n} Processor\_Queue\_Length_i$$
$$+ \beta_1 . \sum_{i=1}^{n} Processor\_Queue\_Length_i \times Process\_Required\_Load_i$$
$$+ \beta_2 . \sum_{i=1}^{n} Processor\_Queue\_Length_i \times Processor\_Current\_Load_i$$

$$+\beta_3 . \sum_{i=1}^{n} Processor\_Queue\_Length_i^2$$
$$+\beta_4 . \sum_{i=1}^{n} Processor\_Queue\_Length_i \times Processor\_Speed_i \quad (10)$$

$$\sum_{i=1}^{n} Processor\_Speed_i \times Processor\_Predicted\_Load_i$$
$$= \beta_0 . \sum_{i=1}^{n} Processor\_Speed_i$$
$$+\beta_1 . \sum_{i=1}^{n} Processor\_Speed_i \times Process\_Required\_Load_i$$
$$+\beta_2 . \sum_{i=1}^{n} Processor\_Speed_i \times Processor\_Current\_Load_i$$
$$+\beta_3 . \sum_{i=1}^{n} Processor\_Speed_i \times Processor\_Queue\_Length_i$$
$$+\beta_4 . \sum_{i=1}^{n} Processor\_Speed_i^2 \quad (11)$$

4 Experiments and results

We have conducted the experiments on a 3 node cluster using Rocks Cluster v5.2 and *Linpack* which is a standard and well-known benchmark for computing environments. Table 4 shows hardware and software specifications of the three nodes.

According to Table 4, *Processor_Speed* is a variable which took one of 3,300, 2,400 or 1,596 values. Also, we obtained *Processor_Queue_Length* using Linux

Table 4 Specifications of the cluster nodes

	Hardware specification	Software specification
Frontend node	4 GB RAM and a 3300 MHz Intel Pentium D	Linux *CentOS 5.2,* HPL-2, *Open MPI* and *ATLAS 3.8.3*
Compute0-0	2 GB RAM and a 2,400 MHz Intel Pentium D	Linux *CentOS 5.2,* HPL-2, *Open MPI* and *ATLAS 3.8.3*
Compute0-1	1 GB RAM and a 1,596 MHz Intel Pentium D	Linux *CentOS 5.2,* HPL-2, *Open MPI* and *ATLAS 3.8.3*

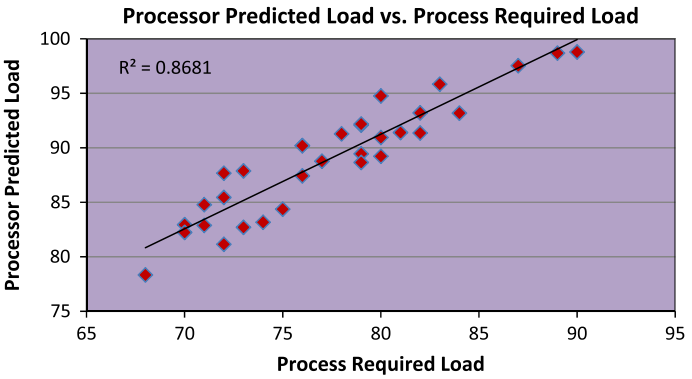

Fig. 2 The relation between predicted load on a processor and required load for a process

commands to count the number of running processes. For example, "*ps aux | wc -l*" counts the number of processes running on a Linux system by any user. There are also other commands that can be used instead. In addition, we had to read the contents of "*/proc/stat*" file at each node to get the values of *Processor_Current_Load,* since Linux does not have any system variable that gives the current load on a processor. Moreover, we have benchmarked the platform twice in identical situations and recorded the required load of processes in the first run and used the data in the second run (i.e., history-based approach) to get data samples for *Process_Required_Load*. Finally, we have recorded processors' loads after running new processes on them so that we have the data for *Processor_Predicted_Load*. Therefore, we have values of all the five variables and we can derive the relation between them using regression analysis.

Figures 2, 3, 4, 5 show the experimental results. For the sake of representation, we have depicted the sample values of variables with regard to *Processor_Predicted_Load*. The R^2 value in each diagram indicates how the two variables are correlated. A higher R^2 value shows a higher correlation between the independent variables and *Processor_Predicted_Load.* For example, an R^2 value of 0.86 in Fig. 2 indicates that *Process_Required_Load* has an 87 % effect on *Processor_Predicted_Load*. This value shows that 87 % of changes in *Processor_Predicted_Load* can be predicted by *Process_Required_Load*.

The horizontal axis in Fig. 2 shows the amount of load that is required by a process. This value is extracted by load balancing mechanism using different approaches, e.g., history-based approach in this research. The vertical axis shows the amount of workload imposed on a processor after running the process on the processor. As it is observed in Fig. 2, the diversity of sample data points shows a high correlation between *Process_Required_Load* and *Processor_Predicted_Load*. This fact is approved by a high value of R^2 that exists in the relation.

The horizontal axis in Fig. 3 shows sample data points of *Processor_Current_Load* taken during executions. This figure shows how current load on a processor influences its future load. The R^2 value quantifies this relation. The value of R^2, in this case, is 0.737 which shows that almost 73 % of changes in the future load is partially caused

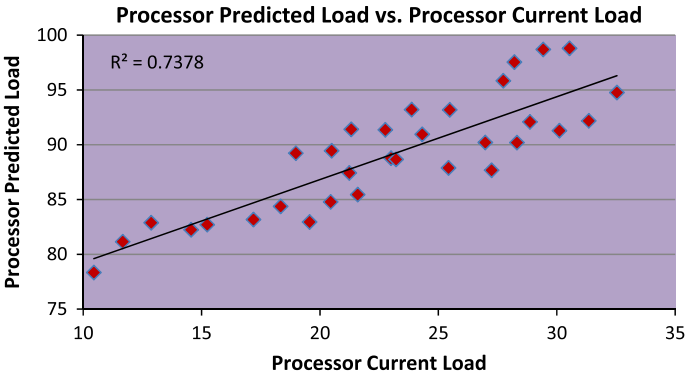

Fig. 3 The relation between processors predicted load and current load on processors

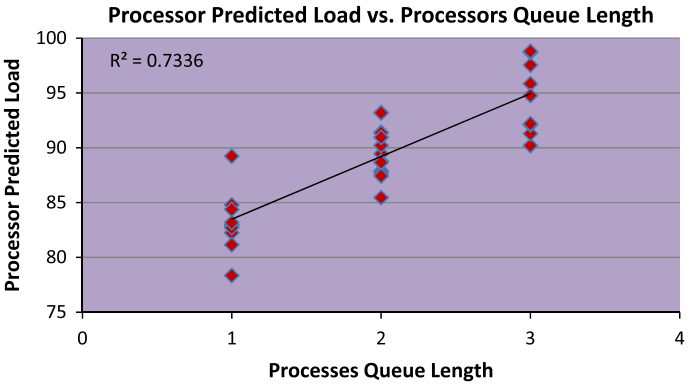

Fig. 4 The relation between processors predicted load and number of processes in their queues

by current load of the processor. This is an approximately high value as it is somehow intuitive in real world.

Figure 4 shows the relationship between number of processes waiting in the queue of processors and their predicted loads. However, each process exerts an additional load on a processor and, therefore, the predicted load on a processor with more number of processes in its queue is more than a processor with fewer waiting processes. This fact is shown by a positive slope in Fig. 4. For example, the predicted load is between 80 and 93 % when there is only one process in the processor's waiting queue while the predicted load is between 90 and 100 % when there are 2 waiting processes in the queue.

Figure 5 shows how the speed of a processor will influence its predicted load. As a rule of thumb, the higher the speed of a processor is, the lower load on the processor will be; considering the same amount of load and a certain time interval. This is due to the fact that a faster processor processes workload faster and, therefore, the amount of workload to process in the future will be less. This fact is, however, indicated by a negative slope in the diagram of Fig. 5. The slope, in fact implies that a slower processor will have more workload to process in the future. For example,

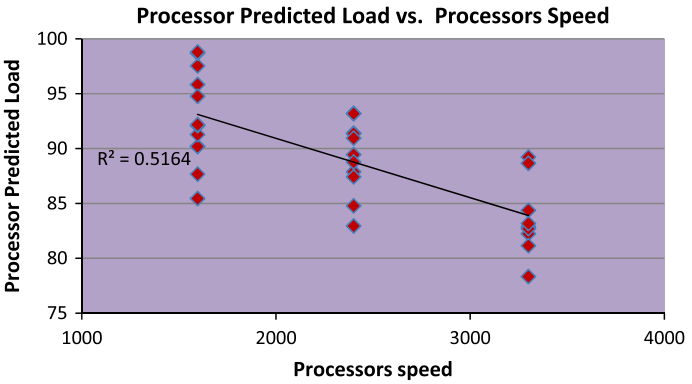

Fig. 5 The relation between processors predicted load and their speeds

Fig. 5 suggests that*compute-0-1* node with a processing speed of 1,596 MHz has more workload to process (approximately 10 % in average) than *frontend* node which has a processing speed of 3,300 MHz.

Nevertheless, we have considered 32 irredundant data samples from the experiments. Therefore, n in Eqs. (7) to (11) is equal to 32. By replacing the experimental data, we reach to the following equalities:

$$\sum_{i=1}^{32} Processor\_Predicted\_Load_i = 2848.89 \tag{12}$$

$$\sum_{i=1}^{32} Process\_Required\_Load_i = 2478 \tag{13}$$

$$\sum_{i=1}^{32} Processor\_Current\_Load_i = 733.36 \tag{14}$$

$$\sum_{i=1}^{32} Processor\_Queue\_Length_i = 63 \tag{15}$$

$$\sum_{i=1}^{32} Processor\_Speed_i = 75252 \tag{16}$$

$$\sum_{i=1}^{n} Process\_Required\_Load_i^2 = 192866 \tag{17}$$

$$\sum_{i=1}^{n} Processor\_Current\_Load_i^2 = 17901.09 \tag{18}$$

$$\sum_{i=1}^{n} Processor\_Queue\_Length_i^2 = 143 \tag{19}$$

$$\sum_{i=1}^{n} Processor\_Speed_i^2 = 191936592 \tag{20}$$

$$\sum_{i=1}^{32} Process\_Required\_Load_i \times Processor\_Predicted\_Load_i = 221457.91 \tag{21}$$

$$\sum_{i=1}^{32} Processor\_Current\_Load_i \times Processor\_Predicted\_Load_i = 66116.54 \tag{22}$$

$$\sum_{i=1}^{32} Processor\_Queue\_Length_i \times Processor\_Predicted\_Load_i = 5717.342 \tag{23}$$

$$\sum_{i=1}^{32} Processor\_Speed_i \times Processor\_Predicted\_Load_i = 6618567.5 \tag{24}$$

$$\sum_{i=1}^{n} Process\_Required\_Load_i \times Processor\_Current\_Load_i = 57466.87 \tag{25}$$

$$\sum_{i=1}^{n} Process\_Required\_Load_i \times Processor\_Queue\_Length_i = 4973 \tag{26}$$

$$\sum_{i=1}^{n} Process\_Required\_Load_i \times Processor\_Speed_i = 5770356 \tag{27}$$

$$\sum_{i=1}^{n} Processor\_Current\_Load_i \times Processor\_Queue\_Length_i = 1574.52 \tag{28}$$

$$\sum_{i=1}^{n} Processor\_Current\_Load_i \times Processor\_Speed_i = 1612607.3 \tag{29}$$

$$\sum_{i=1}^{n} Processor\_Queue\_Length_i \times Processor\_Speed_i = 133668 \tag{30}$$

By substituting Eqs. (12), (13), (14), (15) and (16) in Eq. (7), we reach to Eq. (31):

$$32 \times \beta_0 + 2478 \times \beta_1 + 733.36 \times \beta_2 + 63 \times \beta_3 + 75252 \times \beta_4 = 2848.89 \tag{31}$$

By substituting Eqs. (21), (13), (17), (25), (26) and (27) in Eq. (8), we reach to Eq. (32):

$$2478 \times \beta_0 + 192866 \times \beta_1 + 57466.87 \times \beta_2 + 4973 \times \beta_3 + 5770356 \times \beta_4 = 221457.91 \tag{32}$$

By substituting Eqs. (22), (14), (25), (18), (28) and (29) in Eq. (9), we reach to Eq. (33):

$$733.36 \times \beta_0 + 57466.87 \times \beta_1 + 17901.09 \times \beta_2 + 1574.52 \times \beta_3 + 1612607.3 \times \beta_4 = 66116.54 \quad (33)$$

By substituting Eqs. (23), (15), (26), (28), (19) and (30) in Eq. (10), we reach to Eq. (34):

$$63 \times \beta_0 + 4973 \times \beta_1 + 1574.52 \times \beta_2 + 143 \times \beta_3 + 133668 \times \beta_4 = 5717.342 \quad (34)$$

By substituting Eqs. (24), (16), (27), (29), (30) and (20) in Eq. (11), we reach to Eq. (35):

$$75252 \times \beta_0 + 5770356 \times \beta_1 + 1612607.3 \times \beta_2 + 133668 \times \beta_3 + 191936592 \times \beta_4 = 6618567.5 \quad (35)$$

Eqs. (31) to (35) constitute an equation set with β_0 to β_4 as variables. Since there are 5 variables and 5 equations, the equation set has only one unique answer. By solving this equation set, the values for β_0 to β_4 are determined as the following based on this set of data samples:

$$\beta_0 = 37.4 \quad (36)$$

$$\beta_1 = 0.628 \quad (37)$$

$$\beta_2 = 0.249 \quad (38)$$

$$\beta_3 = 0.007 \quad (39)$$

$$\beta_4 = -0.00115 \quad (40)$$

Finally, by substituting the values of β_0 to β_4 in Eq. (6), we reach to Eq. (41) which shows the relationship between *Processor_Predicted_Load* and *Process_Required_Load, Processor_Current_Load, Processor_Queue_Length* and *Processor_Speed*.

$$Processor\_Predicted\_Load = 37.4 + 0.628 * Process\_Required\_Load + 0.249 * Processor\_Current\_Load + 0.007 * Processor\_Queue\_Length - 0.00115 * Processor\_Speed \quad (41)$$

The value of $R^2$is 0.985 when all variables are considered. This value indicates that almost 98 % of changes in *Processor_Predicted_Load* can be explained by the formula.

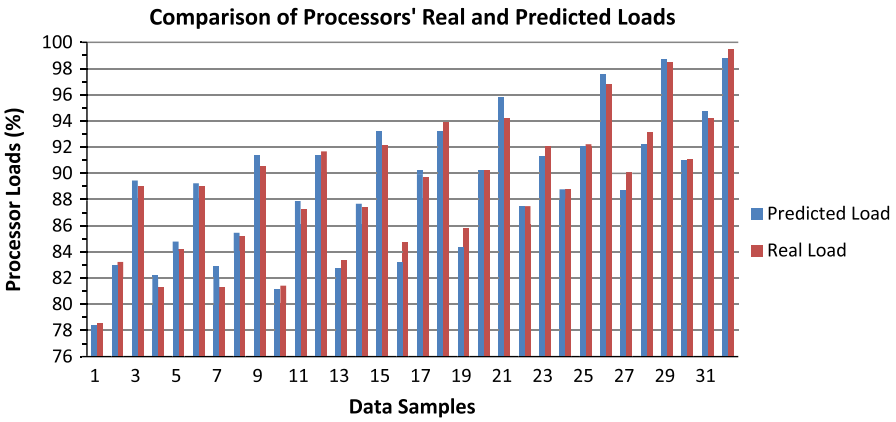

Fig. 6 The difference between real and predicted loads on processors in the data samples

In other words, the formula will generate a correct prediction with a probability of 98 %.

To check the accuracy of predictions, we have recorded the real load on processors after executing processes. Figure 6 demonstrates the differences between real and predicted loads considering the 32 data samples we have used.

As it is shown in Fig. 6, the difference between real and predicted loads is confined to 2 % at most (it occurs at data sample 21). In worst case, if the formula generates a less accurate prediction, the process will pursue its execution on the processor and overloads it. In this situation, load balancing mechanism asks resource discovery mechanism to discover another processor and upon discovery, process migration mechanism migrates the process to the new processor. Since we are not concerned about time critical applications in this paper, the R^2value (i.e., 0.985) has shown to be an acceptable prediction in our research. Nevertheless, a higher probability can be achieved by considering other parameters inside network, dependencies between subtasks, network topologies, global scheduling policies, users' QoS-related variables etc. which we will consider in our future work.

Moreover, Fig. 7 shows how and where the formula will be used in a computing environment. It represents some sample nodes containing load balancing and resource discovery mechanisms in the environment. We have described the communication model between resource discovery and load balancing mechanisms in detail in our previous work, [2].

Figure 7 shows where the formula is used in a computing environment. As shown in Fig. 7, load balancing mechanism extracts the behavior of running processes at each node and when a resource is required, it prepares a request and forwards it to its corresponding resource discovery mechanism. Further, resource discovery mechanism uses the formula to predict the future load on the processor and if it keeps the load under a guaranteed threshold, it will pursue its execution in the host node. The threshold is obtained according to the capacities of each processor. In case the process overloads the processor, resource discovery mechanism will send a request containing *Process_Required_Load* to other nodes. Resource discovery mechanisms at destina-

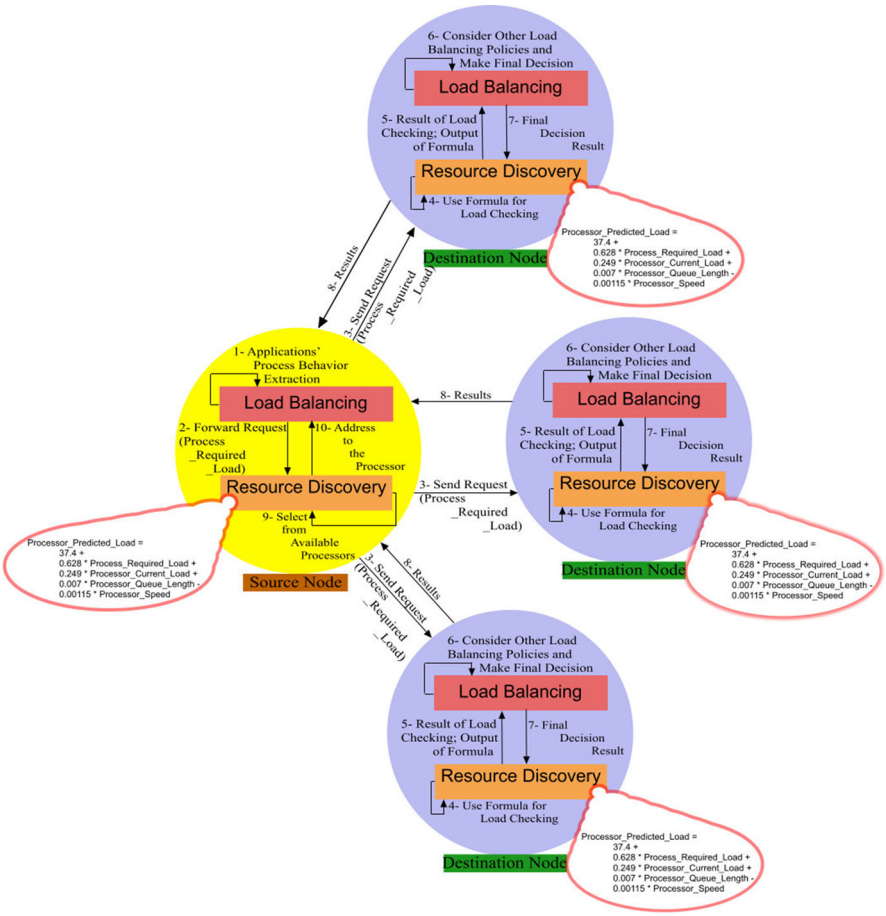

Fig. 7 Two sample nodes communicating with each other in the computing environment

tion nodes use the formula to calculate the future load for load checking. However, the final decision is made by load balancing mechanisms at destination nodes and the results are sent to the source node. Upon reception of results, resource discovery mechanism at source node selects the node with the lowest *Processor_Predicted_Load*.

Therefore, it can be predicted how heavily processors will be used before assigning them new processes using this method and thus their workload will be balanced automatically and no load redistribution will be required.

5 Conclusion and future work

In this paper, we proposed an application and resource aware resource discovery mechanism in computing environments. The goal of this mechanism is to prevent resource overloading through workload prediction. The prediction technique uses the two basic elements of any computing environment, namely resources and applications.

The mechanism predicts how an application's process influences a processor and if it will keep its load under a certain threshold, the process will pursue its execution on the processor. Otherwise another processor will be selected.

The mechanism uses a mathematical formula that parameterizes the behavior of applications and resources' usage patterns to perform predictions. To propose the formula, we used regression analysis which is a statistical method for extracting the relationship among dependent and independent variables based on actual data that are obtained during previous executions of applications. The R^2 value in regression analysis shows how dependent and independent variables are correlated (a higher value indicates a higher correlation). This value, which indicates how much we can rely on predictions, is 98 % in this research with the current set of parameters we have considered. However, we would like to extend the formula in our future work and consider other parameters to make more precise predictions. Also, we would like to investigate and apply this approach to multi- and many-core systems for thread workload prediction and scheduling in our future work.

References

1. Tabbal A, Anderson M, Brodowicz M, Kaiser H, Sterling TL (2011) Preliminary design examination of the ParalleX System from a software and hardware perspective. In: ACM SIGMETRICS, San Jose, pp 81–87
2. Arab MN, Sharifi M (2014) A model for communication between resource discovery and load balancing units in computing environments. J Supercomput 68(3):1538–1555
3. Nitzberg B, Schopf JM, Jones JP (2004) PBS Pro: grid computing and scheduling attributes. Kluwer Academic Publishers, Norwell
4. Meiri E, Barak A (2007) Parallel compression of correlated files. In: Proceedings of IEEE Cluster Computing, Austin, Texas
5. Trunfio P, Talia D, Papadakis H, Fragopoulou P, Mordacchini M, Pennanen M, Popove K, Vlassov V, Haridi S (2007) Peer-to-peer resource discovery in grids: models and systems. J Comput Syst 20(3):864–878
6. Ripeanu M, Foster I, Iamnitchi A (2002) Mapping the Gnutella Network: properties of large-scale peer-to-peer systems and implications for system design. J Internet Comput 6(1):50–57
7. Talia D, Trun P, Zeng J (2007) Peer-to-peer models for resource discovery on grids. J Comput Syst 12(4):864–878
8. Arab MN, Mirtaheri SL, Khaneghah EM, Sharifi M, Mohammadkhani M (2011) Improving learning-based request forwarding in resource discovery through load-awareness. In: International Conference on Data Management in Grid and P2P Systems, Toulouse, pp 73–82
9. Dodonov E, Mello RFD (2010) Novel approach for distributed application scheduling based on prediction of communication events. Future Gener Comput Syst 26(5):740–752