Cluster Comput
DOI 10.1007/s10586-017-0902-8

Dynamic load balancing in distributed exascale computing systems

Seyedeh Leili Mirtaheri[1] · **Lucio Grandinetti**[2]

Received: 13 August 2016 / Revised: 23 March 2017 / Accepted: 2 May 2017

Abstract According to exascale computing roadmap, the dynamic nature of new generation scientific problems needs an undergoing review in the static management of computing resources. Therefore, it is necessary to present a dynamic load balancing model to manage the load of the system, efficiently. Currently, the distributed exascale systems are the promising solution to support the scientific programs with dynamic requests to resources. In this work, we propose a dynamic load balancing mechanism for distributed controlling of the load in the computing nodes. The presented method overcomes the challenges of dynamic behavior in the next generation problems. The proposed model considers many practical parameters including the load transition and communication delay. We also propose a compensating factor to minimize the idle time of computing nodes. We propose an optimized method to calculate this compensating factor. We estimate the status of nodes and also calculate the exact portion of the load that should be transferred to perform the optimized load balancing. The evaluation results show significant improvements regarding the performance by proposed load balancing in compared with some earlier distributed load balancing mechanisms.

Keywords Dynamic load balancing · Distributed systems · Exascale computing · Transition delay · Communication delay · Unpredictable scientific problems

1 Introduction

Nowadays, the high performance computing (HPC) community experiences an astronomical increase in the complexity and sensitivity of scientific and industrial problems [1,2]. Exascale computing is a promising technology to satisfy such computational demands and their challenges [3,4]. Exascale computing systems are capable of at least a billion billion calculations per second. Exascale computing can be designed and developed based on centralized, decentralized or distributed architectures [5]. Distributed exascale computing systems (DECS) are considered as an emerging solution to overcome the complex computing challenges of such scientific programs.

In design and development of next generation high performance computing systems, many challenges should be considered. One of these important challenges is load balancing [6]. Load balancing unit divides the processing load among available computing systems so that processing work is performed in the minimum amount of time. Load balancing can be implemented with software, hardware or a combination of both. According to distributed nature of DECS, load balancing unit should be designed in a completely distributed manner to support large scale distributed systems, free from the constraints of the central units [7]. In distributed load balancing, it is assumed that each node can estimate its own status based on a predefined mathematical model and load balancing mechanism guarantees that the extra load is transferred among computing nodes based on

✉ Seyedeh Leili Mirtaheri
mirtaheri@khu.ac.ir

Lucio Grandinetti
lugran@unical.it

[1] Computer Engineering Department, Kharazmi University, Tehran, Iran

[2] Department of Electronics, Informatics, and Systems, University of Calabria (Unical), Rende, Italy

Published online: 19 May 2017

their computational capabilities, status, and practical limitations.

In the design of the distributed load balancing unit, two major aspects should be considered. First, we need to deal with the complex nature of next generation problems and their unpredictable behaviors. Second, we need to deal with heterogeneous and distributed platforms and their effect in dynamic load balancing. Therefore, many practical factors should be considered in dynamic load balancing. The processing capabilities in command execution and the load status of of nodes should be considered as two important factors in decision making and to distribute the load among the computing nodes [8]. The other important issue is to specify the start time of load balancing mechanism [9]. Different strategies are presented for this issue. In this work, we consider the one-shot strategy for load balancing. In the one-shot policy, the load balancing operation starts when an extra load is imposed on the network. Another important group of effective parameters in load balancing mechanisms are related to the interconnection of computing nodes in the distributed platform [10]. The important effective parameter from this group is the communication delay between the nodes that is neglected in the research literature.

In this paper, we propose a distributed load balancing mechanism for distributed exascale computing systems. Distributed load balancing means that all the nodes make a decision by themselves while cooperate with their neighbor processing systems [11]. There are two strategies for load distribution among the nodes, named sender-initiated strategy and receiver-initiated strategy [12]. We use sender-initiated strategy based on a distributed platform. We focus to find the optimum amount of processing load for each node while the idle time of the computing nodes in the system is minimized. We present a model for dynamic load balancing that calculates the accurate amount of extra load in each node by considering the proposed distributed model and also the load transfer delays. In this paper, we consider the effect of all practical parameters including the communication and load transfer delays which highly affects the performance of load balancing mechanism. In proposed method, a parameter named Compensating Factor is modeled and calculated to transfer the accurate portion of the load to neighbor nodes by considering the communication delay. We also suggest an optimum method to calculate the value of this factor to transfer the optimum amount of load.

The rest of this paper is organized as follows. Section 2 reviews some related works and different load balancing models with concerned aspects of this work. In Sect. 3 the major issues of proposed load balancing model are mentioned. In Sect. 4 we describe the proposed distributed load balancing model. Evaluation results are presented in Sect. 5 and Sect. 6 concludes the paper.

2 Related works

In recent years, dynamic load balancing area has attracted a lot of attention and many researchers have contributed handling the load of the system at runtime. In [13] authors propose a dynamic parallel scheduling algorithm for load balancing. The dynamicity in load balancing systems is considered to overcome two issues, the dynamic behavior of the problems and underlying platforms. An adaptive load balancing for iterative computation on heterogeneous systems is proposed in [14]. Distributed systems are divided into two categories based on the interconnection of nodes, namely fully connected and partially connected [15,16]. In fully connected systems, each node is directly connected to all other nodes in the system while in partially connected distributed systems, each node has a direct connection only to its neighbor nodes. In practice, implemented distributed systems are partially connected networks and therefore, we focus on this type of systems.

For distributing the load in dynamic load balancing mechanisms, several strategies are proposed that could be categorized into sender-initiated and receiver-initiated strategies. In sender-initiated strategy, the processor that is currently over-loaded, look forward to send a portion of its load to another computing node [17]. In receiver-initiated strategy, the under-loaded processor initiates the transaction by sending a message to other nodes [18]. Depending on the load balancing algorithm and network topology, both receiver and sender based strategies are implemented. However, the task transfer cost under receiver-initiated policies is significantly higher than sender-initiated policies and therefore, in general, sender-initiated policies provide better performance. In [19], a sender-initiated strategy is proposed that diffuses work to nearby neighbors in order to balance the domains and also causes the load of all processors to be even. In this method, each processor acts independently, distributing some of its load to low-load neighbors. Balancing is performed by each of the processors when it receives a load update from a neighbor stating that its load is below a threshold [19].

Load balancing mechanism is designed based on different concerns such as response time, utilization, throughput and etc. The main concern of many load balancing algorithms is to decrease the response time or equivalently the total execution time by efficient use of available resources [20]. Load balancing and data distribution are major issues in large machines to overcome the challenges of exascale computing in supporting new unpredictable scientific problems. In [21], authors propose a load balancing mechanism for massive parallel computations based on sparse grid combination techniques [21]. In [22], the concept of a resource sharing barrier (RSB) for load balancing is introduced. This barrier allows process groups to wait on other processes work and actively contribute to their completion. Actually, RSB

enables groups that have completed their own work and lend their processor resources temporarily. However, dividing the system or processes to groups is in contradiction with the fully distributed algorithms.

Exascale computing will require efficient job management and load balancing to overcome the next generation computing issues. One of these issues is related to the nature of the problems. For instant, data intensive problems have experienced an astronomical increase. In balancing the load of systems with computing-intensive tasks, usually, tasks are randomly migrated from heavy-loaded schedules to idle ones. For data-intensive applications, where tasks are dependent and task execution involves processing a large amount of data, blind task migrating yields poor data-locality and incurs significant data-transferring overhead. Some research works are performed to a present load balancing model in terms of this issue. In [23], authors present an optimized load balancing algorithm by organizing the tasks in queues based on task data size and location. They implement their technique in MATRIX. However, they do not specify the portion of the load that should be transferred by forming unpredictable events in runtime. Supporting a mixture of applications in different domains, such as traditional large-scale HPC, the ensemble runs, and the fine-grained many-task computing is one of the issues in exascale computing systems. Delivering high performance in resource allocation, scheduling and launching for all types of jobs, have encouraged some researchers to develop a distributed workload manager that is directly extended from the centralized production systems [24]. The proposed method in [24], named Slum++, employs multiple controllers so that each one manages a partition of computing nodes and participates in resource allocation through resource balancing techniques. They propose a monitoring-based weakly consistent resource stealing technique to achieve resource balancing in distributed HPC. The main focus of Slum++ is on resources. Unlike [24], in this paper our focus is relevant to events on behalf of the program and exact estimation of the extra load should that be transferred.

Another important issue in exascale computing is related to energy consumption and some researches present load balancing mechanisms by considering this issue. Authors in [25], investigate a self-organized critical approach for a dynamic load-balancing. They propose a model based on the Bak–Tang–Wiesenfeld sandpile: a cellular automaton that works in a critical regime at the edge of chaos. The results of their experiments show that when the arrival rate of tasks is equal to the processing power of the system, the system has a serious attractor. Based on this issue, to maximize the utilization of resources, they assume that the processing units can be turned on and off depending on the state of the workload. The results show that the energy consumption of the system, in general, is minimized without compromising the quality of service. The other research in terms of this issue is performed in [26]. They present a method to model energy and thermal behavior of computing systems. This model is very efficient for simulation of workload and resource management in computing systems and to evaluate the system performance, energy consumption, and energy-efficiency. The accuracy results are evaluated by comparing the values calculated based on these models against the measurements results obtained on real hardware. In [27], authors investigate migration of applications for HPC by deriving respective requirements for specific field of applications. They sketch example scenarios demonstrating its potential benefits. They present a prototype migration mechanism enabling the seamless migration of MPI processes in HPC systems.

Although, in mentioned researches many practical factors are considered, there still exist unresolved issues. Most of the research works in literature are performed in a centralized manner. In distributed mechanisms, some new challenges such as load transition should be considered. taking into account the unpredictable behavior of exascale computing systems, we should evaluate the effect of delays in load balancing. Two types of delays should be considered in load balancing algorithms, named transition delay and communication delay. The transition delay is addressed in literature while the communication delay is neglected. Ignoring the communication delay is a logical assumption when the processing power of nodes is low or moderate. However, by recent improvements in computational capabilities of processing systems, communication delays should be included in system model as well as load transition delay. In this paper, unlike many other works, we assume that the network is partially connected and computing nodes only communicate with their neighbor nodes. We model the load balancing problem considering the communication delay and propose an optimized method to compensate this delay.

3 Load balancing system model

The system model for load balancing in distributed exascale computing systems is described in this section.

Exascale computing is a computing system with the capability of at least one exaFLOPS or a billion billion calculations per second. This computing system should satisfy the future concerns with the approach of hardware, system software, and application [28]. The architecture of this system can be centralized or distributed. One of the main challenges of exascale computing is to develop a runtime system that supports fully autonomic management of resources, including adaptive policies that identify and react to load imbalances and the intermittent loss of resources. Therefore, partitioning and load balancing algorithms for exascale systems and usage of many-core libraries are common con-

cerns. The dynamic load balancing model should potentially address two major challenges of DECS, namely distributed resources and unpredictable behavior of new generation of scientific programs.

3.1 Load status model

In the load balancing model with centralized architecture, the number of tasks in each node in the time period of Δt is related to some parameters mentioned as follows:

- The number of assigned tasks to node i at the moment of t named (L_i^{old})
- The number of tasks in node i at the moment of $t + \Delta t$ named (L_i^{new})
- The number of new assigned tasks to node i in time period of Δt named (J_i)
- The number of executed tasks in node i in time period of Δt named (C_i)
- The number of passed tasks from node i to node j in t_l^i time ($t < t_l^i < \Delta t$) that it is the result of load balancing mechanism and we show it by ($L_j\,(i)$).
- The number of passed tasks from node j to node i that passed at the time $t - \tau_{ji}^k$ but because of communication delay arrived to node i in time period of $t+\Delta t$. We named this tasks ($L_i\,(j)$).

Finally based on the introduced parameters, the following mathematical model is presented to calculate the number of available tasks at the time period of $t + \Delta t$ in node i:

$$L_i^{new} = L_i^{old} - C_i + J_i - \sum_{j \neq i} L_j(i) + \sum_{j \neq i} L_i(j) \qquad (1)$$

In Eq. (1), by calculating L_i^{new}, each node can estimate its status and will be a member of two following sets, I, and I^c. I is a set of nodes with a positive load, and I^c is a set of nodes with the negative load. The nodes with positive load have less computing power than the considered load for them and a portion of their load should be transferred to other computing nodes. The nodes with negative load have more capacity to execute the tasks and it is possible to assign more load to them.

Essentially, this load balancing mechanism can be introduced for estimating $L_j\,(i)$s, amount of transferred load from a node with positive load, i, to the nodes with negative load, j.

3.2 Load balancing start time

The other important question in load balancing operations is "when is the best time to start the load balancing operation?". It should be noted that execution of load balancing operation increases the response time of the assigned program. Therefore, the load balancing operation should be performed in the suitable start time. Different policies are proposed for load balancing start time. Based on the number of load balancing execution perspective, we can divide them into two groups, named one-time and reassignment policies. Considering one-time policy, when the load is assigned to the nodes, it is not possible to reassign that load to other computing nodes. However, in reassignment policy, there is the possibility of revision in the amount of assigned load to each node. In this paper, we use one-time policy.

The one-time policy group includes another policy named one-shot. In one-shot policy, load balancing operation is performed when a new load enters the network. It means that the system works normally before the entrance of a new load. When the new load enters, the system status changes and it is necessary to execute load balancing operation.

3.3 Transferring load portion

The other important challenge of proposed load balancing model is the portion of the load that should be transferred from nodes with a positive load to nodes with a negative load. In this work, we propose a model to specify this amount of load with the goal of balancing the received load among all nodes and all nodes. We try to transfer the load so that when nodes start to execute their own portion of the load, the idle time of whole system is minimized and all computing nodes finish their execution at the same time. In this model two factors are calculated, task execution speed up and task transferring speed up. Task execution speed up is calculated by determining the number of tasks executed in that node and task transferring speed up is calculated by determining the required time to transfer the tasks.

According to stochastic nature of task execution in the computing systems, it seems that finding a statistical distribution to model the process of commands execution in the nodes is essential. We assume that the behavior of nodes to execute commands on the average sequential execution of commands and performing an averaged action is available. However, transient behavior of the nodes in executing the instructions should be modeled appropriately. Poisson distribution is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time and/or space if these events occur with a known average rate and independently of the time since the last event. The Poisson distribution can also be used for the number of events in other specified intervals such as distance, area or volume. A discrete random variable X is said to have a Poisson distribution with parameter $\lambda > 0$, if, for $k = 0, 1, 2, \ldots$, the probability mass function of X is given by Eq. (2).

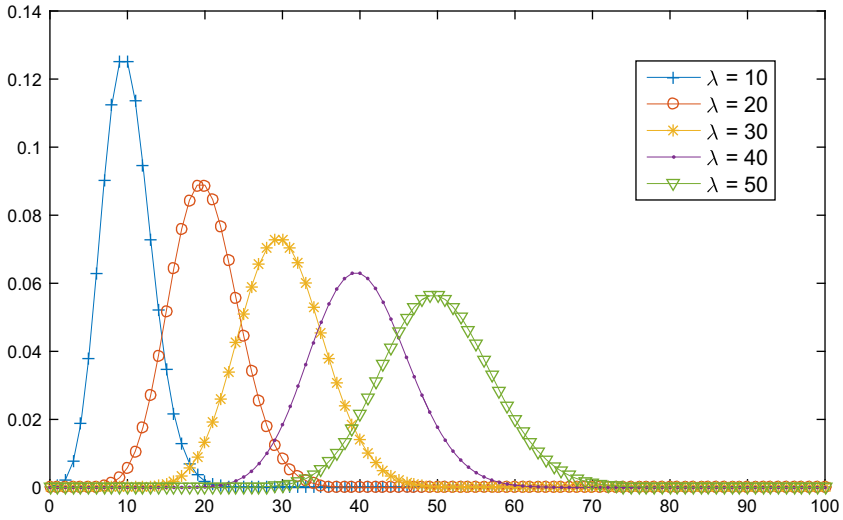

Fig. 1 The Poisson distribution probability function

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{2}$$

where e is Euler's number ($e = 2.71828.$) and $k!$ is the factorial of k. Poisson distribution probability function diagram is shown in Fig. 1. To show the effectiveness of the used Poisson distribution, a simple program for adding $1+1$ is written that is executed by a PC with a processor with core2dou 2.4 GHz and 2 GB of RAM. According to the results, the computer runs x commands per second. This number is obtained by calculating the average speed of execution (λ) in 15 min time period.

After estimating this number, based on some tests, a number of executed commands in the same computer at intervals of 1 microsecond are measured. Repeating the tests in the time period of 2 s, some charts are obtained from the moment behavior of executing the commands on that computer. In addition, by using the Poisson function, some sample from Poisson distribution is produced. Figure 2 shows these two diagrams, as it can be seen, diagrams related to the behavior of commands execution in the computer have a good agreement with the produced diagrams of Poisson distribution.

As mentioned before, the issue of load balancing is transferring additional load from one overloaded node to low load nodes to minimize the total execution time. Its clear that the process of load balancing imposes delay to the system. According to the high speed of commands execution in todays computers, the communication delay is an effective factor that should not be neglected. For modeling the delay of transferred tasks from one node to another, using an exponential distribution is accepted in engineering science. The probability density function of an exponential distribution is presented in Eq. (3).

$$P(x = t) = \lambda e^{-\lambda x} \tag{3}$$

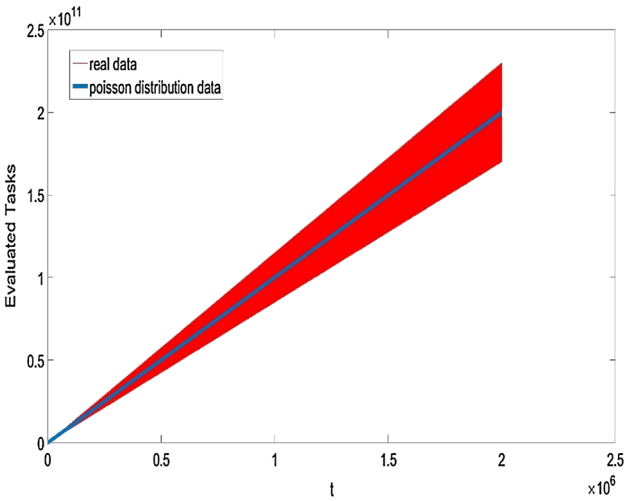

Fig. 2 Comparison of numbers of executed commands in some real test and numbers of executed commands in the same period of time by Poisson distribution

It should be noted that to consider the transferring delay, the transmission line behavior should also be modeled. In this work, we proposed a parameter named Compensating Factor to consider transferring delay in load balancing decisions.

4 Proposed dynamic load balancer in fully distributed platform

Since the goal of this research is modeling the effective parameters and proposing an optimized load balancing mechanism, we should consider the practical limitations of the distributed exascale computing system in the model. Therefore, it is appropriate to model computer communication system with a graph. In this graph, each node shows a computing node and the edges show the connection of these nodes.

$$L_i^{new} = L_i^{old} - C_i + J_i - \sum_{j \in N_i} L_j(i) + \sum_{j \in N_i} L_i(j) \quad (4)$$

Considering a distributed system, the Eq. (1) should be amended. This relationship will be modified as Eq. (4). L_i^{new} calculates the number of available tasks at the time period of Δt in node i. By calculating L_i^{new}, each node can estimate its status. In this work, each node will be a member of two sets, I, and I^c based on its status. Essentially, this load balancing mechanism can be introduced to estimate $L_{il}(r)$, the amount of transferred load from a node with positive load, r, to a node with negative load, l.

Applying the distributed features and using the one-shot mechanism in the proposed load balancing, each overloaded node acts as a central node and starts to balance its load by using gathered information about the status of neighbor nodes. To manage the extra load, the status information of neighbor nodes is used. It means that each node is aware of current load status and also speed up in executing commands of its neighbor nodes. It is obvious that communication of nodes introduces a delay to the system and is not persistent. Therefore, passing time, the gathered information about the status of the nodes are outdated. In this situation, it is possible to estimate the status information of neighbor nodes based on their speed up in executing the commands. For example, if δt seconds passes from information receiving time of node i and the speed up of node i is λ_i, it is possible to estimate the load of node i by using $L_i - \lambda_i \Delta t$.

Indeed, to convert the centralized load balancing into distributed format, we should add the index of the node with an extra load in the equations to show that the calculated result is based on the view of this node. To present the proposed equations and to calculate the status of computing nodes, we define some parameters as follows. Moreover, it is necessary to note that, the defined parameters have the index i, that shows the value of the parameter from the perspective of it node.

- $\mathrm{I} = \left\{i | L_i^{ex} < 0\right\}$: Set of nodes with a negative load.
- $\mathrm{I}^c = \left\{j | L_j^{ex} > 0\right\}$: Set of nodes with a positive load.
- $\mathrm{I}_i = \left\{l | l \in N_i, L_{il}^{ex} < 0\right\}$: Set of nodes with a negative load from perspective of ith node.
- $\mathrm{I}_i^c = \left\{r | r \in N_i, L_{ir}^{ex} > 0\right\}$: Set of nodes with a positive load from perspective of ith node.
- L_{ir}^{ex}, L_{il}^{ex}: The calculated extra load in rth node and lth node from perspective of ith node
- $R = |\mathrm{I}_i|$: Number of neighbor nodes to it node that have a negative load.
- $L = \left|\mathrm{I}_i^c\right|$: Number of neighbor nodes to ith node that have a positive load.
- λ_{rl}: Load transferring speed up from rth node to lth node.
- λ_r, λ_l: Command execution speed up in rth node and lth node.
- T_{ir}, T_{il}: Command execution time in rth node and lth node from perspective of ith node.
- T_r, T_l: Command execution time in rth node and lth node.
- $L_{il}(r)$: The amount of transferred load from rth node to lth node from perspective of ith node.
- $L_l(r)$: The transferred load from rth node to lth node.
- $P_{il}(r)$: Extra load transition coefficient from rth node to lth node from perspective of ith node.
- $P_l(r)$: Extra load transition coefficient from rth node to lth node.
- $K_{il}(r)$: Compensating Factor for transferring the load from rth node ton lth node from perspective of ith node.
- $K_l(r)$: Compensating Factor for transferring the load from rth node to lth node.
- $t_{il}(r)$: The receiving time of the load transferred from rth node to lth node ($L_{il}(r)$) that it is calculated by ith node.
- $t_l(r)$: The receiving time of the load transferred from rth node to lth node ($L_l(r)$).
- $t_{il}(r') = \left\{t_{il}(r) | r' \in \mathrm{I}_i, t_{il}(r'-1) < t_{il}(r')\right\}$: Set of $t_{il}(r)$ that it is just ascending.
- $L_{il}(r') = \left\{L_{il}(r) | r' \in \mathrm{I}_i, t_{il}(r'-1) < t_{il}(r')\right\}$: Set of $L_{il}(r)$s that they are arranged according to $t_{il}(r')$s.
- $t_l(r') = \left\{t_l(r) | r' \in \mathrm{I}, t_l(r'-1) < t_l(r')\right\}$: Set of $t_l(r)$s that it is just ascending.
- $L_l(r') = \left\{L_l(r) | r' \in \mathrm{I}, t_l(r'-1) < t_l(r')\right\}$: Set of $L_l(r)$s that they are arranged according to $t_l(r')$s.

Now, assume that the extra load is entered to ith node, this node should act as a central node and balance the load by calculating L_{ir}^{ex} and $L_{il}(r)$ based on the following equations:

$$L_{il}^{ex} = L_l - \frac{\lambda_l}{\sum_{k \in N_i} \lambda_k} \sum_{k \in N_i} L_k \quad (5)$$

$$L_{il}(r) = p_{il}(r) L_{ir}^{ex} \quad (6)$$

$$p_{il}(r) = \frac{L_{il}^{ex}}{\sum_{m \in \mathrm{I}_i} L_{im}^{ex}} \quad (7)$$

In which $\mathrm{I}_i = \left\{l | l \in N_i, L_{il}^{ex} < 0\right\}$. It is obvious that all the nodes connected to the node i can be divided into two groups, I_i and I_i^c that are defined as:

$$\mathrm{I}_i = \left\{l | l \in N_i, L_{il}^{ex} < 0\right\} \quad (8)$$

$$\mathrm{I}_i^c = \left\{r | r \in N_i, L_{ir}^{ex} > 0\right\} \quad (9)$$

Considering Fig. 3 as the connection of distributed nodes, assume that the extra load is entered to node 5 and the nodes 1, 2 and 4 are the neighbors of node 5. Nodes 1 and 2 have a negative load and nodes 4, 5 have a positive load. In this situation, we will have Fig. 4 as the working graph. If the nodes 1 and 2 have negative load and nodes 4 and 5 have positive load, our equation will be: $L_{51}(4) = \left\lfloor p_{51}(4) L_{54}^{ex} \right\rfloor$,

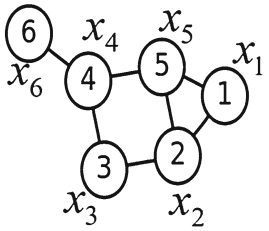

Fig. 3 The graph of the distributed system model

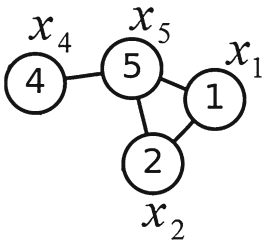

Fig. 4 Equivalent graph for the case where the extra load is entering the fifth node

$L_{51}(5) = \lfloor p_{51}(5) L_{55}^{ex} \rfloor$, $L_{52}(4) = \lfloor p_{52}(4) L_{54}^{ex} \rfloor$ and $L_{52}(5) = \lfloor p_{52}(5) L_{55}^{ex} \rfloor$, in which:

- $p_{51}(5) = p_{51}(4) = \frac{L_{51}^{ex}}{L_{51}^{ex} + L_{52}^{ex}}$
- $p_{52}(5) = p_{52}(4) = \frac{L_{52}^{ex}}{L_{51}^{ex} + L_{52}^{ex}}$

It should be noted that the first index in $L_{il}(r)$ and L_{il}^{ex} means the value of $L_l(r)$ and L_l^{ex} are from perspective of ith node. This means that from perspective of node 5, the system has a model in the form of Fig. 4 and not in the form of Fig. 3. After calculating extra load, ith node calculates the transferred load from rth node to lth node. With this assumption that the load transferred speed up from rth node to lth node is λ_{rl}, it takes $t_{il}(r)$ second to transfer load $L_{il}(r)$, to destination that can be calculated by Eq. (10).

$$t_{il}(r) = \frac{L_{il}(r)}{\lambda_{rl}} \tag{10}$$

For better description of the system, we sort the entered load to lth node ($L_{il}(r)$) based on their receive time ($t_l(r)$) and two following sets are defined:

$$L_{il}(r') = \left\{ L_{il}(r) | r' \in \mathrm{I}_i, t_{il}(r'-1) < t_{il}(r') \right\} \tag{11}$$

$$t_{il}(r') = \left\{ t_{il}(r) | r' \in \mathrm{I}, t_{il}(r'-1) < t_{il}(r') \right\} \tag{12}$$

Figure 5 shows the time model of events that happens in load balancing operation in lth node from perspective of ith node. This diagram shows the start time of load balancing operation and $\tilde{L}_{il}(r')$ shows the amount of load of ith node before reaching the new load and we have:

$$\tilde{L}_{il}(r') = \left[\tilde{L}_{il}(r'-1) + L_{il}(r'-1) - \lambda_l \Delta t_{il}(r') \right]^+ \tag{13}$$

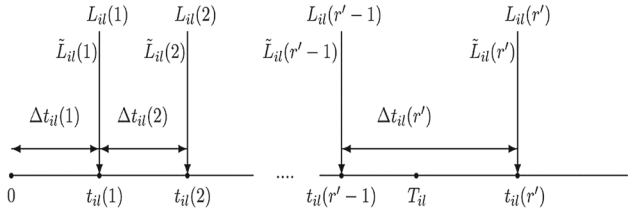

Fig. 5 Time model of load balancing in node l

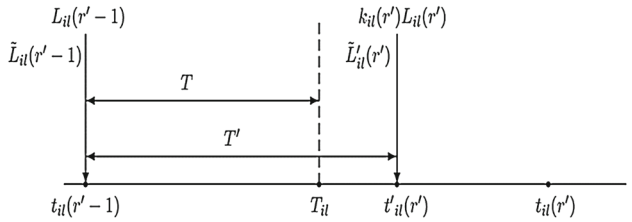

Fig. 6 The time period between $t_{il}(r'-1)$ and $t_{il}(r')$

in which:

$$\Delta t_{il}(1) = t_{il}(1) \tag{14}$$

$$\Delta t_{il}(r') = t_{il}(r') - t_{il}(r'-1) \tag{15}$$

Because of load transferring delay, two events may occur. First, the source node or the load transferring node finishes its processing before the transfer of extra load to the destination node. Happening of this event is infeasible, because, as mentioned in the previous section, all the presented equations for load balancing grantees that load of all nodes finishes at the same time. Second, assume that the lth node finishes its load execution before receiving a load from r'th node. This might happen and the finish time will be equal to:

$$T_{il} = t_{il}(r'-1) + \frac{\tilde{L}_{il}(r'-1) + L_{il}(r'-1)}{\lambda_l} \tag{16}$$

Actually, in this situation, part of the system resources capacity is lost. In other word, lth node remains idle in time period of T_{il} to $t_{il}(r')$, while in this time period this node is able to run $\left(t_{il}(r') - T_{il}\right)\lambda_l$ commands. For better understanding of this issue, the time period of $t_{il}(r'-1)$ to $t_{il}(r')$ is shown separately in Fig. 6. We should minimize this delay time by choosing optimum portion of load to be transferred by considering the delay time in our calculations. Compensating Factor is proposed to calculate optimum amount of load.

4.1 Compensating factor

The compensating Factor proposed in this work is named $k_{il}(r')$ and $k_{il}(r') \leq 1$. By considering this factor, $L_{il}(r')$ should be changed to $k_{il}(r')L_{il}(r')$. It decreased transfer time of load to lth node and therefore, the transferred load is

received at the time $t'_{il}(r')$. The compensating factor is equal to:

$$k_{il}(r') = \frac{\lambda_{r'l} t'_{il}(r')}{p_{il}(r') L^{ex}_{ir'}} = \frac{\lambda_{r'l}\left[t_{il}(r'-1) + T'\right]}{p_{il}(r') L^{ex}_{ir'}} \tag{17}$$

It should be noted that the Eq. (17) is a general form to calculate the compensating factor. This is the way of choosing $t'_{il}(r')$ that specify the value of $k_{il}(r')$. On the other words, its possible to gain new value for $k_{il}(r')$ by considering the influence of different parameter in choosing $t'_{il}(r')$. In this work, a method is proposed to calculate an optimum value for $k_{il}\left(r'\right)$. We also present the analysis of this method in this paper. Since $p_{il}(r')L^{ex}_{ir'}$ is transferred at $t_{il}(r') = t_{il}(r'-1) + \Delta t_{il}(r')$ time, we can write Eq. (18) as follows:

$$k_{il}(r') = \frac{\lambda_{r'l}\left[t_{il}(r'-1) + T'\right]}{\lambda_{r'l}\left[t_{il}(r'-1) + \Delta t_{il}(r')\right]} = \frac{t_{il}(r'-1) + T'}{t_{il}(r'-1) + \Delta t_{il}(r')} \tag{18}$$

It is clear that, the maximum value of $k_{il}(r')$ is one, in this situation, $L_{il}(r')$ will be transferred, that it is equal to specified value in last load balancing operation. Larger values than the optimum choice of $k_{il}(r')$, causes the node to be in idle mode. However, the load balancing operation should act in such a way that all nodes finish their processing at the same time and minimizes the idle time. According to Eq. (18), for $k_{il}(r') = 1$ we will have $T' = \Delta t_{il}(r')$. The minimum value of $k_{il}(r')$ causes $t'_{il}(r') = T_{il}$. It should be noticed that the goal of calculating a value smaller than one for $k_{il}(r')$ is minimizing the idle time of lth node and by choosing $t'_{il}(r') = T_{il}$ it is not necessary to minimize $k_{il}(r')$. In this situation, according to the Eq. (18), we have $T' = \Delta t_{il}(r')$. Therefore, an approximated value of $k_{il}(r')$, T' and $t'_{il}(r')$ are calculated as follows:

$$\frac{t_{il}(r'-1) + T}{t_{il}(r'-1) + \Delta t_{il}(r')} \le k_{il}(r') \le 1 \tag{19}$$

$$T \le T' \le \Delta t_{il}(r') \tag{20}$$

$$T_{il} \le t'_{il}(r') \le t_{il}(r') \tag{21}$$

The parameters $k_{il}(r')$, T' and $t'_{il}(r')$ are dependent and therefore, choosing each one affects the choice of others. In continue, we will present how to determine these values. As mentioned before, the node r', transfers a portion of its load to lth node as a load balancing mechanism. The amount of this load is equal to $p_{il}(r')L^{ex}_{ir'}$. By transferring this load, the remained extra load in r'th node will be distributed to other nodes in set of I_i, with this constraint that the finish time of load execution in all computing nodes should be the same. By calculating a value except one? for $k_{il}(r')$, amount of $k_{il}(r')p_{il}(r')L^{ex}_{ir'}$ load from $p_{il}(r')L^{ex}_{ir'}$ load will be transferred to lth node and amount of $\left[1 - k_{il}(r')\right] p_{il}(r')L^{ex}_{ir'}$ load remains in r'th node. It causes r'th node to execute its load later than other nodes. This delay is equal to execution time of the mentioned amount of load remained in lr'th node. This time is depended to load execution speed up in r'th node and we show it by $\lambda_{r'}$ that is equal to:

$$\begin{aligned}
\frac{\left[1 - k_{il}(r')\right] p_{il}(r')L^{ex}_{ir'}}{\lambda_{r'}} &= \frac{p_{il}(r')L^{ex}_{ir'}}{\lambda_{r'}} - \frac{k_{il}(r')p_{il}(r')L^{ex}_{ir'}}{\lambda_{r'}} \\
&= \frac{\lambda_{r'l}}{\lambda_{r'}} t_{il}(r') - \frac{\lambda_{r'l}}{\lambda_{r'}} t'_{il}(r') = \frac{\lambda_{r'l}}{\lambda_{r'}}\left[t_{il}(r') - t'_{il}(r')\right] \\
&= \frac{\lambda_{r'l}}{\lambda_{r'}}\left[t_{il}(r'-1) + \Delta t_{il}(r') - t_{il}(r'-1) - T'\right] \\
&= \frac{\lambda_{r'l}}{\lambda_{r'}}\left[\Delta t_{il}(r') - T'\right]
\end{aligned} \tag{22}$$

To calculate $k_{il}(r')$, we propose to set equal idle time for node l and node r'. The idle time of lth node is equal to $T' - T$, based on the Fig. 6, and idle time of lr'th node based on the Eq. (22) is calculated as follows:

$$\frac{\lambda_{r'l}}{\lambda_{r'}}\left[\Delta t_{il}(r') - T'\right] = T' - T \tag{23}$$

By simplifying the equation Eq. (23), we have:

$$\begin{aligned}
\frac{\lambda_{r'l}}{\lambda_{r'}}\Delta t_{il}(r') + T &= T'\left(1 + \frac{\lambda_{r'l}}{\lambda_{r'}}\right) \\
\frac{\lambda_{r'l}}{\lambda_{r'}}\Delta t_{il}(r') + T &= T'\left(\frac{\lambda_{r'} + \lambda_{r'l}}{\lambda_{r'}}\right)
\end{aligned}$$

Therefore, we have:

$$T' = \frac{\lambda_{r'l}}{\lambda_{r'} + \lambda_{r'l}}\Delta t_{il}(r') + \frac{\lambda_{r'}}{\lambda_{r'} + \lambda_{r'l}} T \tag{24}$$

Value of $k_{il}(r')$ based on both T' and Eq. (18) will be:

$$\begin{aligned}
k_{il}(r') &= \frac{t_{il}(r'-1) + \frac{\lambda_{r'l}}{\lambda_{r'} + \lambda_{r'l}}\Delta t_{il}(r') + \frac{\lambda_{r'}}{\lambda_{r'} + \lambda_{r'l}} T}{t_{il}(r'-1) + \Delta t_{il}(r')} \\
&= \frac{\frac{\left[\lambda_{r'} + \lambda_{r'l}\right] t_{il}(r'-1)}{\lambda_{r'} + \lambda_{r'l}} + \frac{\lambda_{r'l}}{\lambda_{r'} + \lambda_{r'l}}\Delta t_{il}(r') + \frac{\lambda_{r'}}{\lambda_{r'} + \lambda_{r'l}} T}{t_{il}(r'-1) + \Delta t_{il}(r')} \\
&= \frac{\frac{\lambda_{r'l}}{\lambda_{r'} + \lambda_{r'l}}\left[t_{il}(r'-1) + \Delta t_{il}(r')\right] + \frac{\lambda_{r'}}{\lambda_{r'} + \lambda_{r'l}}\left[t_{il}(r'-1) + T\right]}{t_{il}(r'-1) + \Delta t_{il}(r')}
\end{aligned}$$

Finally, we have:

$$k_{il}(r') = \frac{\lambda_{r'l}}{\lambda_{r'} + \lambda_{r'l}} + \frac{\lambda_{r'}}{\lambda_{r'} + \lambda_{r'l}}\left[\frac{t_{il}(r'-1) + T}{t_{il}(r'-1) + \Delta t_{il}(r')}\right] \tag{25}$$

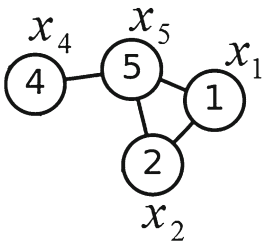

Fig. 7 The network topology of nodes

In Eq. (24), if the speed up of load transferring from r'th node to lth node goes to infinity, it means $\lambda_{r'l} \to +\infty$, then we will have $k_{il}(r') \to 1$ and when $\lambda_{r'l} \to 0$ then we will have $k_{il}(r') \to \frac{t_{il}(r'-1)+T}{t_{il}(r'-1)+\Delta t_{il}(r')}$ that it is equal to the same period intended for $k_{il}(r') \to 1$ in Eq. (19).

5 Evaluation of the proposed model

We evaluate the performance of our proposed algorithm using computer simulations which are coded as a toolbox for MATLAB software. We consider a computer network composed of multi-systems as a connected graph. As mentioned earlier, we assume that the distributions of task execution speed up in each node and the communication delay between nodes have Poisson and Exponential distribution, respectively. Load balancing with one-shot policy in a multi-computer network depends on different parameters including task execution speed up in each node, delay of load transferring between two nodes, the initial load of each node, network graph model that defines the connection between nodes, external load insertion time instant and the node facing the external node.

In this section, we define simulation scenarios with controlled variation of mentioned parameters to evaluate the effect of load-transfer delay and to evaluate the performance of proposed model. In the first simulation, we consider a network with 6 nodes, each having 300 initial tasks and all with the same processor speed. Considering one-shot policy, we assume the external load as 1000 tasks implied to node 1. The network topology is depicted in Fig. 7.

First of all, we assume that all nodes are capable of performing 500 tasks per second and transferring 100 tasks from one node to another per second. We assume that the insertion time of the external load to the system is at $t = 0.1$. If we do not use the load balancing, the execution process of tasks in the network is shown in Fig. 8.

It should be noted that the instantaneous load in each node is depicted, therefore a linear decay instead of random or Poisson distribution is observed. As seen in Fig. 8, the external load is inserted to node 1 at time $t = 0.1$. Although, due to lack of load balancing, this node does not transfer its external load to the neighbor nodes (node 2 and 5 according to topology) and the finish execution time will be at $t = 2.6$s and all node will be ideal almost for 2 s. The total ideal time of the system will be about 10 s.

In the second simulation, we perform load balancing. However, we do not consider the load-transfer delay according to what is presented in [19]. The execution process of tasks in the network is presented in Fig. 9.

In Fig. 9, each node performs the load balancing without considering the load transfer delay. According to simulated network topology, node 2 and 5 are the neighbors of node

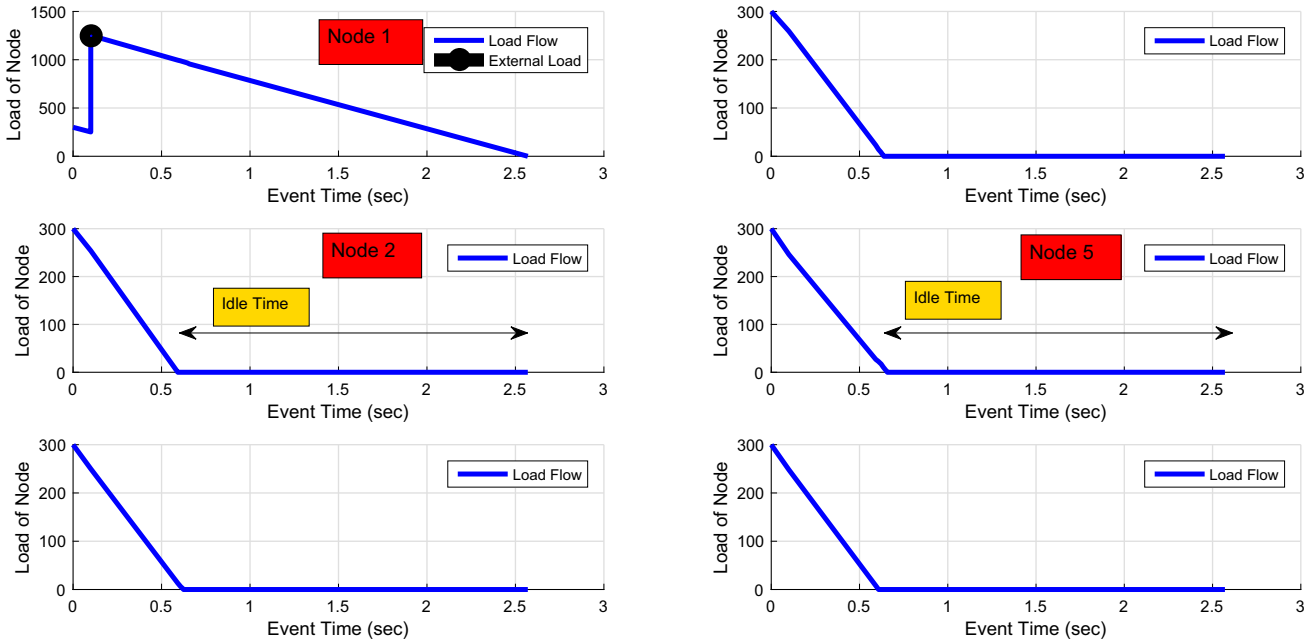

Fig. 8 The task execution process versus time in network when load balancing is not performed

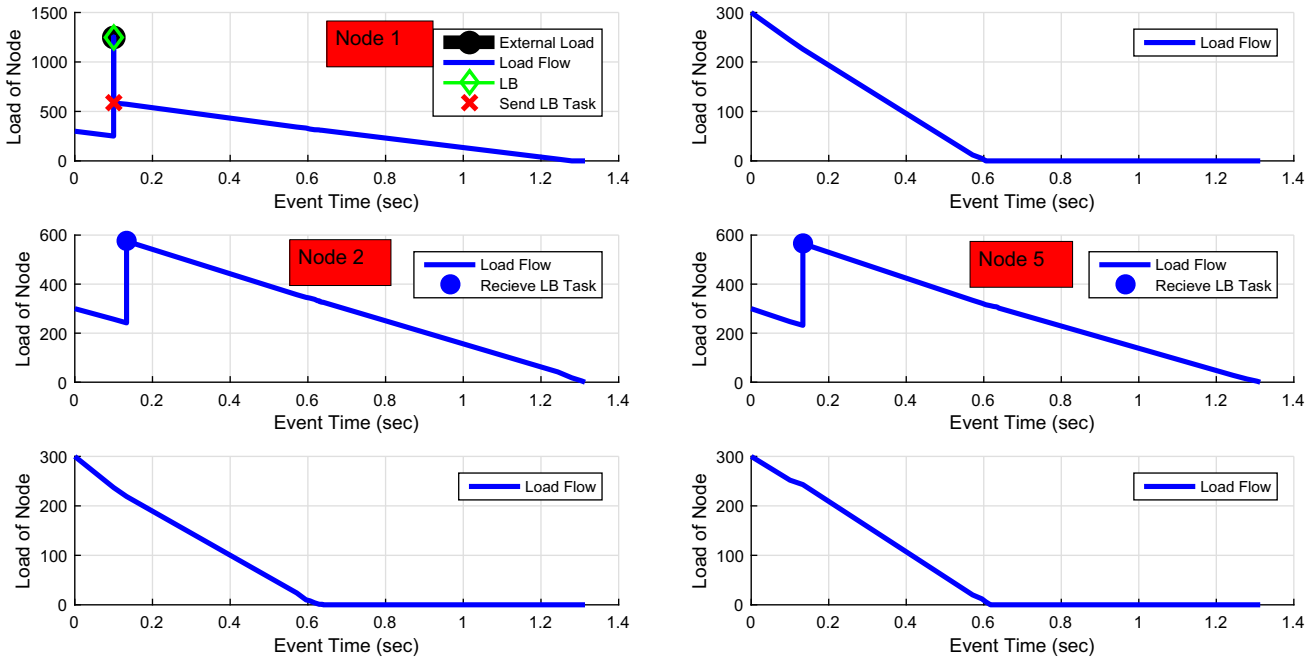

Fig. 9 The task execution process versus time in the network when load balancing is performed without load transfer delay considerations

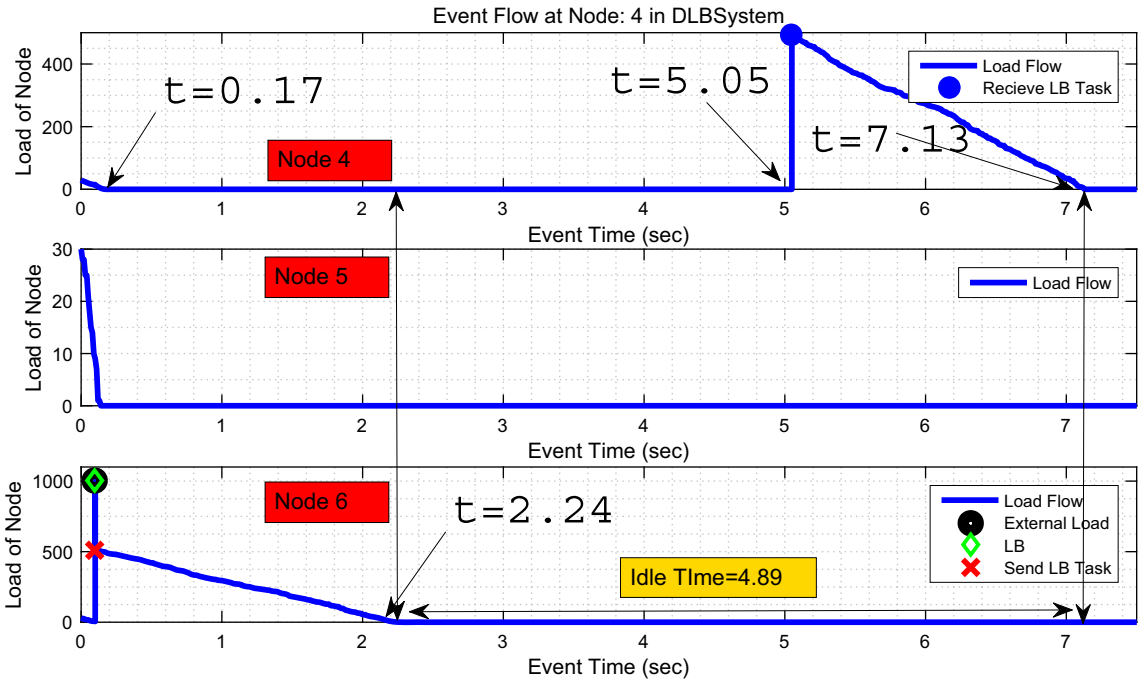

Fig. 10 The effect of load transfer delay on idle time of systems

1. Therefore, the external load is transferred to node 2 and 5. Due to load balancing operation, the execution time of node 1, 2 and 5 is almost the same. If our network was fully connected (all nodes were directly connected to each other), the execution time of all nodes after load balancing was the same. As depicted in Fig. 9, it is obvious that the total execution time is reduced after load balancing operation and the finish time of load execution in that system is $t = 1.32$s that shows about 50% decrease. In this scenario, almost 660 task is transferred to Node 5 and Node 2 and the total ideal time of the system is 2.34 s that shows 76.6% improvement. As mentioned before, in performed simulation, we did not consider the load-transfer delay from node 1 to node 2 and node 5 that is 0.033 s in load balancing. Even considering this delay

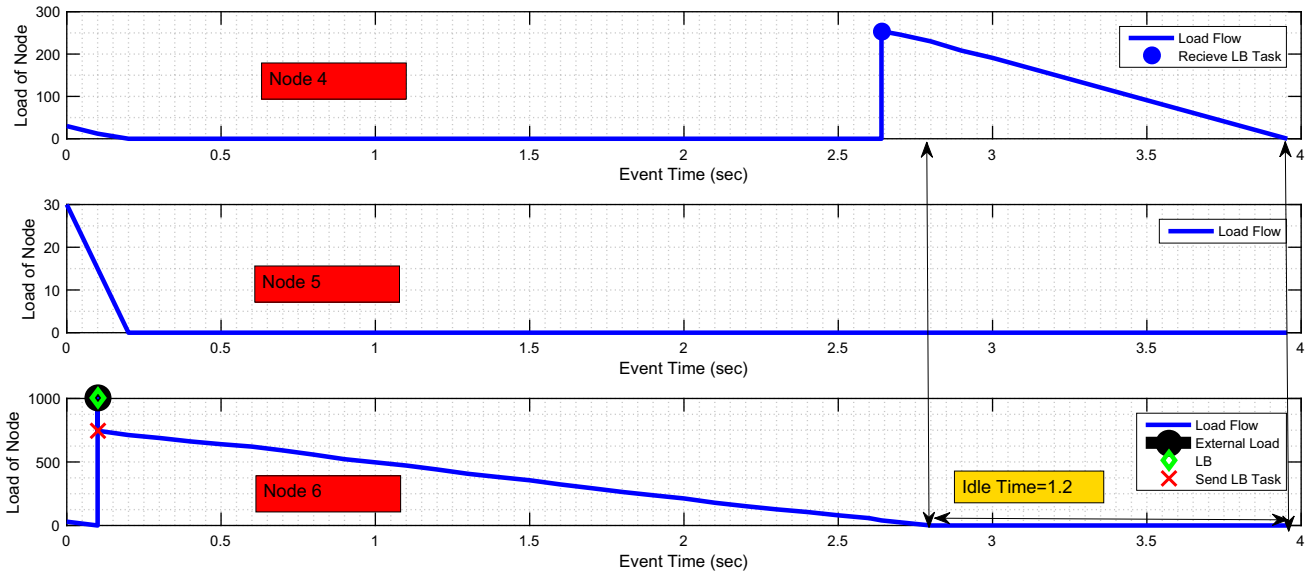

Fig. 11 The performance evaluation of proposed method on total task execution time

in load balancing, the results would be the same. Because the neighbor nodes (node 2 and 5) had initial load and these recourses were fully used until the arrival of external load. To illustrate the effect of load transfer delay, we consider another scenario where all nodes have 300 initial tasks. The load execution speed is considered to be 200 tasks per second for each node and 100 tasks per second are transferred from one node to another. In this scenario, we assume that the external load is inserted into node 6 having node 5 as its sole neighbor. The results are depicted in Fig. 10.

As depicted in Fig. 10, node 4 performs all assigned tasks at $t = 0.17$ and it is in standby (idle) state until $t = 5.05$ when the load is reached to node 4. The load balancing operation is performed in a way that all nodes finish their tasks together. The standby time of node 4 causes extra delay in the system and node 4 finishes its assigned tasks at $t = 7.13$ while node 6 finishes the tasks at $t = 2.24$. According to the definition of total execution time, the execution time for assigned tasks is equal to $t = 7.13$. Therefore, it is necessary to compensate for this load transfer delay and to prevent the idle time in the system that is 4.89 s. Now, we evaluate the performance of this network with our proposed methods. Here, we evaluate the performance of our proposed method in reducing the total execution time. The simulation results are depicted in Fig. 11.

In Fig. 11, we consider the load transfer delay and apply our proposed load balancing method. Node 6 transfers 311 tasks to node 4 instead of 495 tasks in the previous section. This causes a significant reduction of standby time for node 4 (3 s instead of 5 s in previous simulation). Therefore, the total task execution time of system reduces to 4.8 s and it shows about 67% improvement and the ideal time of the system decrease to 1.2 that shows 75.4% improvement.

6 Conclusion

In this work, we studied the problem of dynamic load balancing in distributed exascale computing systems. We proposed a distributed model in which each node calculates the number of available tasks at the specific time period. Based on the estimated status, the nodes were divided into two groups, nodes with negative load and the nodes with a positive load. Practically, our proposed load balancing mechanism is performed to estimate the extra load in each node that should be transferred by considering the transition and communication delays. We defined a parameter named compensating factor that helped us to calculate the optimum amount of load that should be transferred by taking into account the load transfer and communication delays. We proposed an optimized algorithm to calculate the compensating Factor to improve the performance of proposed load balancing mechanism, especially in the systems that the transferring delay is notable. The evaluation results illustrated the substantial improvement about 76.6% in the efficiency of the proposed load balancing in compared with the case where no load balancing mechanism was applied. In another scenario, we considered the effect of transition delay and evaluated our proposed model. In the same experiment conditions, and using compensating factor, the results showed 75.4% improvement in compared with the case that no compensating factor was considered in load balancing.

References

1. DOE Workshop Report (2014).: Software Productivity for Extreme-Scale Science. Rockville

2. Mirtaheri, S.L., Khaneghah, E.M., Grandinetti, L., Sharifi, M.: A mathematical model for empowerment of Beowulf clusters for exascale computing. In: High Performance Computing and Simulation (HPCS), 2013 International Conference on, pp. 682-687. IEEE, Helsinki (2013)
3. Dongarra, J.: International Exascale Software Project Roadmap (Draft 1/27/10 5: 08 PM) (2009)
4. strm, J.A., Carter, A., Hetherington, J., Ioakimidis, K., Lindahl, E., Mozdzynski, G., Westerholm, J.: Preparing scientific application software for exascale computing. In: International Workshop on Applied Parallel Computing, pp. 27–42. Springer, Berlin (2012)
5. Wang, K., Kulkarni, A., Lang, M., Arnold, D., Raicu, I.: Exploring the design tradeoffs for extreme-scale high-performance computing system software. IEEE Trans. Parallel Distrib. Syst. **27**(4), 1070–1084 (2016)
6. Kogge, P., Bergman, K., Borkar, S., Campbell, D., Carson, W., Dally, W., Hill, K.: Exascale computing study: technology challenges in achieving exascale systems (2008)
7. Qin, X., Jiang, H., Manzanares, A., Ruan, X., Yin, S.: Dynamic load balancing for I/O-intensive applications on clusters. ACM Trans. Storage (TOS) **5**(3), 9 (2009)
8. Reddy, H.: Performance Evaluation of Static and Dynamic Load-Balancing Schemes for a Parallel Computational Fluid Dynamics Software Application (Fluent) Distributed Across Clusters of Heterogeneous Symmetric Multiprocessor System. IBM Red Book, 6609 Carriage Drive Colleyville, TX 76034 (2004)
9. Mohamed, N., Al-Jaroodi, J.: Delay-tolerant dynamic load balancing. In: High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on, pp. 237–245. IEEE (2011)
10. Llanes, A., Cecilia, J.M., Snchez, A., Garca, J.M., Amos, M., Ujaldn, M.: Dynamic load balancing on heterogeneous clusters for parallel ant colony optimization. Cluster Comput. **19**(1), 1–11 (2016)
11. Langer, A.: An optimal distributed load balancing algorithm for homogeneous work units. In: Proceedings of the 28th ACM international conference on Supercomputing, pp. 165–165. ACM (2014)
12. Alam, T., Raza, Z.: An adaptive threshold based hybrid load balancing scheme with sender and receiver initiated approach using random information exchange. Practice and Experience, Concurrency and Computation (2016)
13. Mahafzah, B.A., Jaradat, B.A.: The hybrid dynamic parallel scheduling algorithm for load balancing on Chained-Cubic Tree interconnection networks. J. Supercomput. **52**(3), 224–252 (2010)
14. Martnez, J.A., Almeida, F., Garzn, E.M., Acosta, A., Blanco, V.: Adaptive load balancing of iterative computation on heterogeneous nondedicated systems. J. Supercomput. **58**(3), 385–393 (2011)
15. Ybenes, P., Escudero-Sahuquillo, J., Garca, P.J., Quiles, F.J.: Straightforward solutions to reduce HoL blocking in different Dragonfly fully-connected interconnection patterns. J. Supercomput. **72**(12), 1–23 (2016)
16. Mirtaheri, S.L., Sharifi, M.: An efficient resource discovery framework for pure unstructured peer-to-peer systems. Comput. Netw. **59**, 213–226 (2014)
17. Balasangameshwara, J., Raju, N.: Performance-driven load balancing with a primary-backup approach for computational grids with low communication cost and replication cost. IEEE Trans. Comput. **62**(5), 990–1003 (2013)
18. Domanal, S.G., Reddy, G.R.M.: Load Balancing in Cloud Environment using a Novel Hybrid Scheduling Algorithm. In: 2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), pp. 37–42. IEEE (2015)
19. Dhakal, S., Hayat, M.M., Pezoa, J.E., Yang, C., Bader, D.A.: Dynamic load balancing in distributed systems in the presence of delays: a regeneration-theory approach. IEEE Trans. Parallel Distrib. Syst. **18**(4), 485–497 (2007)
20. Mkel, A., Siikavirta, S., Manner, J.: Comparison of load-balancing approaches for multipath connectivity. Comput. Netw. **56**(8), 2179–2195 (2012)
21. Heene, M., Kowitz, C., Pflger, D.: Load Balancing for Massively Parallel Computations with the Sparse Grid Combination Technique. In: PARCO, pp. 574–583. (2013)
22. Arafat, M.H.: Runtime Systems for Load Balancing and Fault Tolerance on Distributed Systems (Doctoral dissertation, The Ohio State University), (2014)
23. Wang, K., Zhou, X., Li, T., Zhao, D., Lang, M., Raicu, I.: Optimizing load balancing and data-locality with data-aware scheduling. In: Big Data (Big Data), 2014 IEEE International Conference on, pp. 119–128. IEEE (2014)
24. Wang, K., Zhou, X., Qiao, K., Lang, M., McClelland, B., Raicu, I.: Towards scalable distributed workload manager with monitoring-based weakly consistent resource stealing. In: Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing, pp. 219–222. ACM (2015)
25. Laredo, J.L.J., Guinand, F., Olivier, D., Bouvry, P.: Load Balancing at the edge of chaos: how self-organized criticality can lead to energy-efficient computing. IEEE Trans. Parallel Distrib. Syst. **28**(2), 517–529 (2016)
26. Pitek, W., Oleksiak, A., Da Costa, G.: Energy and thermal models for simulation of workload and resource management in computing systems. Simul. Modell. Pract. Theory **58**, 40–54 (2015)
27. Pickartz, S., Lankes, S., Monti, A., Clauss, C., Breitbart, J.: Application migration in HPCA driver of the exascale era?. In: High Performance Computing & Simulation (HPCS), 2016 International Conference on, pp. 318–325. IEEE (2016)
28. Alowayyed, S., Groen, D., Coveney, P.V., Hoekstra, A.G.: Multiscale Computing in the Exascale Era. arXiv preprint arXiv:1612.02467 (2016)

Seyedeh Leili Mirtaheri is faculty member of Electrical and Computer Engineering Department in Kharazmi University. She is researching on next generation high performance computing systems. He worked as a researcher in Excellent Center of High Performance Computing for Parallel and Distributed Processing in Italy. Her research interests are in the areas of distributed and parallel systems, peer-to-peer computing, cluster computing, mathematics, and scientific computing. She has worked on Distributed Systems and done a number of successful industrial experiments in these areas. She Received First award of Inventions at National Science Foundation Invention Festival, 2011, IUST (Iran University of Science and Technology)'s Awards for Excellence in Researching in 2009, Second Level Reward of National Science Foundation in PHD duration, 2009, First Award for presenting CSharifi: Kernel Level Cluster Management System Software", at the Khwarizmi Young Awards, 2008 and Grant of Excellent Researcher of National Science Foundation, 2008 and Iranian Organization of Scientific and Industrial Research appreciation to cooperating and presenting "A Cluster Management System Software" at the Khwarizmi International Awards, 2007.

Lucio Grandinetti Professor Emeritus at the Department of Computer Engineering, Electronics, and Systems of University of Calabria (UNICAL), Italy. At the same University he holds the position of Vice Rector. He graduated from the University of Pisa, Italy and the University of California at Berkeley. He has been a post-doc fellow at University of Southern California, Los Angeles and Research Fellow at the University of Dundee, Scotland. He was a member of the IEEE Committee on Parallel Processing, and European Editor of the book series of MIT Press on Advanced Computational Methods and Engineering. Currently he is member of the Editorial Board of four international journals. He is author of many research papers in well-established international journals and Editor or co-Editor of several books on algorithms, software, applications of Parallel Computing, HPC, Grids, Clouds. He has been recipient and scientific leader of many European-Commission-Funded projects since 1993 (e.g. Molecular Dynamics Simulations by MPP Systems, EUROMED, HPC Finance, WADI, BEINGRID). Currently he is Director of the Centre of Excellence on HPC established at the University of Calabria by the Italian government; Co-managing director of a Supercomputing Centre jointly established by the University of Calabria and NEC Corporation.