CCF Transactions on High Performance Computing (2023) 5:390–415
https://doi.org/10.1007/s42514-022-00134-8

REGULAR PAPER

ExaLB: a mathematical framework for load balancing to support distributed exascale computing environments

Faezeh Mollasalehi[1] · Ehsan Mousavi Khaneghah[1] · Amirhosein Reyhani Showkatabadi[2] · Seyed Alireza Seyednejad[2] · Faeze Gholamrezaie[3]

Received: 19 February 2022 / Accepted: 28 December 2022 / Published online: 13 February 2023

Abstract
The dynamic and interactive nature of Distributed Exascale Computing System leads to a situation where the load balancer lacks the proper pattern for the solution. In addition to analyzing and reviewing the dynamic and interactive nature and its effect on load balancing, this article introduces a framework for managing load balancing that does not need to study the dynamic and interactive nature. This framework proposes a mathematical scheme for the functionality of load-balancing elements and redefines its functions and components. The redefinition makes it possible to determine the constituent parts of the framework and their functionality without the need to analyze the dynamic and interactive nature of the system. The proposed framework can manage and control dynamic and interactive events by reviewing changes in the functionality of resources, the pattern of data collection to execute processes related to the load balancer, and a Scalable tool. In addition to performing the load balancer's functionality, our framework can continue to function under dynamic and interactive events in distributed exascale systems. On average, this framework has a 43% improvement, unable to respond to dynamic and interactive requests.

Keywords Distributed exascale computing system · Load balancing · Functionality · Resource description · Request description

1 Introduction

In high performance computing systems, the load balancer is responsible for adjusting computing processes, demands, and computing capabilities of the existing resources (defined in the computing environment) (Wang et al. 2014; Ghomi et al. 2017; Jyoti and Shrimali 2020; Amelina et al. 2015). Therefore, the load balancer should be able to manage computing processors in the existing system so that (A) the response time of any processor should exceed what is expected and (B) no computing resource should be idle while executing a scientific or practical application (Domanal et al. 2014; Thakur and Goraya 2017). The load balancer needs precise information about the requirements of computing processors and features of the existing resources in the computing system to achieve this (Khaneghah et al. 2018). If the element has accurate information about computing processors and resources, it can adjust between processor requirements and computing resources and features (Mondal et al. 2016; Mondal et al. 2017; Mondal et al. 2016). Proper adjustment between the two mentioned characteristics leads

✉ Ehsan Mousavi Khaneghah
EMousavi@Shahed.ac.ir

Faezeh Mollasalehi
Faezeh.mollasalehi@shahed.ac.ir

Amirhosein Reyhani Showkatabadi
ah.reyhani@ut.ac.ir

Seyed Alireza Seyednejad
Seyednejad1993@ut.ac.ir

Faeze Gholamrezaie
faeze.gholamrezaie@shahed.ac.ir

[1] Department of Computer Engineering, Faculty of Engineering, Shahed University, Tehran, Iran

[2] College of Engineering, School of ECE, University of Tehran, Tehran, Iran

[3] Department of Computer Science, Shahed University, Tehran, Iran

to a better response time for computation processes, which increases the system's performance (Kołodziej et al. 2014; Qureshi et al. 2014).

Load balancer conventionally has to collect data, decide about the resources and computing processes, and also process transmission (Khan et al. 2017; Pate Ahmadian et al. 2018; Jain and Saxena 2016). The element must describe the status of the processes, process requirements to perform these tasks, and system resources, using specific metrics (Bok et al. 2018; Rao et al. 2003). The element could perform the mentioned tasks in the case of proper and exact definitions for descriptions of processes and resources.

Based on these metrics, the load balancer decides whether it can continue executing the process in a processing element. In its view, the continuation of performing a process depends on the process's requirements, conditions, and time limitations and whether the processing elements can meet them (Mousavi Khaneghah et al. 2018). If these conditions are not met, the component uses process migration to transfer the process from one processing element to another (Khan et al. 2017; Rathore et al. 2020; Rathore and Chana 2016). Suppose the system's processing elements cannot respond to the process demands. In that case, the aspect calls on resource discovery to find the processing capable of meeting the requirements of the process (Pourqasem 2018).

The load-balancer establishes the structure of responding patterns created during design at the runtime (Khaneghah et al. 2018). In traditional computing systems, the selected mechanisms for load balancing, such as Cluster or Grid, are designed to make them compatible with the structure of response control and management related to process demands (Ramezani et al. 2014; Heidsieck et al. 2019; Teylo et al. 2017; Khaneghah and Sharifi 2014). System designers know the computing process's needs in such traditional computing systems. Based on this information, proceed to define mechanisms for load balancing or resource discovery (Pate Ahmadian et al. 2018; Milani and Navimipour 2016).

The primary assumption in designing a traditional load balancer such as a Cluster or Grid is based on the fact that nothing violates the structure of the designed response during the execution of a scientific application (Khaneghah and Sharifi 2014). Based on this, if a process or processes require computing resources, the load balancer tries to use process migration or resource discovery to choose the resource which can respond to the process (Pate Ahmadian et al. 2018). Since resource allocation is based on the load balancer's mechanism, it has been considered part of the policy related to the structure of responding patterns in scientific applications (Khaneghah and Sharifi 2014).

The occurrence of dynamic and interactive events in executing computing processes in DECS leads to processes performing operations or requests that were not considered in the primary responding pattern (Khaneghah and Sharifi 2014; Milani and Navimipour 2016; Fiore et al. 2018; Gharb et al. 2019; Dongarra et al. 2011). This event causes the system not to be capable of responding based on the preliminary responding design during the execution of scientific applications. In case of such events, the load balancer must be able to analyze the event and create a proper responding structure based on its nature to continue executing the scientific application (Wang et al. 2016a). These events might need different response patterns (Shahrabi et al. 2018; Alowayyed et al. 2017; Innocenti et al. 2017). The load balancers should be able to create the response structure for such events in an appropriate period. Due to changes in the characteristics of resources and the response structure of processes, traditional systems either cannot be used in DECS or provide the minimum performance possible (Alowayyed et al. 2017; Innocenti et al. 2017; Khaneghah et al. 2018).

In distributed Exascale computing systems, the occurrence of dynamic and interactive events from the point of view of the system manager means the formation of a particular type of request related to the process (or processes) that has not been analyzed in the primary response structure. The occurrence of dynamic and interactive events causes the state of requests in the computing system to change in a way unknown to the load balancer. The load balancer should either stop the system, and the system designer explains the mentioned situation for the load balancer to execute the request, or it must manage the said request based on a mechanism at the time of execution.

When a dynamic and interactive event occurs in a large-scale distributed system, in addition to changing the status of requests in the system, as described in the previous paragraph, it can cause a change in the working process of the distributed load balancer as a central unit of the activities in a large-scale distributed system. The functional nature of the distributed load balancer in computing systems, both traditional and Distributed Exascale systems, is based on collecting information and using a decision-making mechanism for redistributing the load, in addition to creating a mechanism for implementing the program in the shortest time and achieve the system's goal. The load balancer collects the information related to the system (or a part of the system), and based on the analysis obtained by a mechanism in the load balancer, it performs redistribution activities. The way the processes are executed by the operating system, keeping in mind that this is a distributed system, causes the set of activities related to the distributed management unit not to be executed automatically and continuously, instead these activities are performed at different times considering the corrections of the information in the previous step. The consideration of the corrections to the information in the previous stage is conventionally established in traditional computing systems, and the existence of benchmarking

tools and data structure information related to the process and, most importantly, the definition of specific situations to carry out the activities of the load balancer, makes it possible to decide on the continuation or discontinuation of it.

In Distributed Exascale systems, at any moment of the execution of distributed load balancer's activities, dynamic and interactive events can affect the load balancer's functioning and sub-activities. A dynamic and interactive event may occur in the large-scale distributed system, which causes (a) the distributed load balancer not to be able to decide on the assumption about the correctness of the information of the previous stage and the status of the large-scale distributed system regarding the correctness of the information to be unknown to the load balancer. (b) The status of the beneficiary elements in the activities of the distributed load balancer should be changed in such a way that the set of activities of the load balancer becomes invalid in terms of redistribution and the mechanism used. (c) The mechanism used by the load balancer cannot manage and redistribute the load for the large-scale distributed computing system. (d) The descriptive and benchmarking indicators used by the distributed management unit become invalid after the dynamic and interactive event occurs.

In each or all of the mentioned situations, the distributed management unit in Distributed Exascale systems must be able to manage the impacts of dynamic and interactive events on its functionality. This article, introducing the ExaLB, presents a mathematical model to manage caused by dynamic and interactive events on the load balancer in distributed Exascale computing systems without the need to stop the execution of system activities.

This paper introduces a mathematical framework to describe the load balancer's performance in DECS. The descriptive function of the load distribution management element in distributed computing systems is investigated to achieve this framework. Based on the review and analysis of the mentioned descriptive function, the framework describing the load balancer operation in the distributed Exascale systems obtained in this article makes it possible to analyze the concept of the event that leads to the call of the load balancer in the distributed computing system. Did Consider this issue in this article, the events leading to the call of the load balancer are considered in two categories, formal events, and dynamic and interactive events. The mentioned two classifications and the analysis of the management of the mentioned events can be used as a criterion for the analysis of dynamic events and how to manage them in the middle of the proposed mathematical framework for the load balancer, taking into account the conditions and limitations imposed by dynamic and interactive events.

2 Related works

Currently, many scientific applications are developed for DECS. In this system, computing resources are connected in an autonomous, transparent, and integrated manner and usually support scalability (Jiang 2016). In high performance computing and distributed systems, a load balancer is used to distribute the tasks fairly to execute the application in the shortest time possible (Chatterjee and Setua 2015).

There are multiple definitions proposed for the scalability of distributed systems. In Domanal et al. (2014), scalability is considered a function of changes in the system performance when a new processing element is added. In Mirtaheri et al. (2013), the system's scalability is introduced as a function with parameters such as the cost of adding a new computing element and system performance when it is being expanded. Determining the relation between load balance and the system's scalability is highly important if this relation is not selected correctly. Scalability would not be capable of improving the performance and also adds to the load balancer's execution time, leading to decreased system efficiency.

In Distributed Exascale systems, scheduling existing tasks to improve performance is vital (Chatterjee and Setua 2015; Mukherjee et al. 2016). In Mukherjee et al. (2016), a method for efficient load balancing in large-scale systems is introduced in which equivalent Markov models describe parallel servers. This computing system uses a threshold-based load-balancing scheme.

The challenges of DECS consist of managing processes in parallel, the procedure of executing scientific applications, usable processing power, flexibility, and scalability (Wang et al. 2016b). Many computing system management schemes, such as load balancing mechanisms, are designed based on centralized paradigms and have to define a centralized server. This state creates challenges for DECS, such as scalability and single-point-of-failure problems (Wang et al. 2016b). In Wang et al. (2016b), a classification for system management in DECS is proposed that considers the proportion between the server's response time and the client's tolerance. In this categorization, there is a discussion about what pattern of reliability is created by system scalability in the procedure of execution. Based on Wang et al. (2016b), using an architecture based on distributed systems to support extreme parallelism, covering delay time, and creating mechanisms for a reliable and scalable load balancer, are necessary.

The nature of scientific applications requires DECS to use computing systems to reduce the response time and discover the laws governing natural phenomena (http://www.deep-project.EU; Reylé et al. 2016). Fields such as

human brain simulation, weather simulation, fluid engineering computations, simulating superconduction in high temperatures, and earthquake imaging are some of the applications that require DECS (http://www.deep-project.EU; Reylé et al. 2016). The nature of these applications (in contradiction to traditional computing applications) is a way in which, during the execution of processes, dynamic and interactive requests are made by them that their structures are not considered during the design (Lieber et al. 2016; Mirtaheri et al. 2014; Straatsma et al. 2017; Dongarra et al. 2014). This event causes DECS to use dynamic mechanisms for load balancers (Milani and Navimipour 2016; Khaneghah 2017). Therefore, computing systems should have the flexibility and compatibility with such variant conditions in these applications to achieve optimal system performance.

In Alowayyed et al. (2017), there has been a discussion on how to improve the capabilities of multiscale computing systems to execute scientific applications that require Petascale and DECS. In Jeannot et al. (2016), a dynamic mechanism for the load balancer in the Charm++ application is proposed.

The authors of Mirtaheri and Grandinetti (2017) have also proposed a distributed dynamic mechanism for the load balancer in DECS. This mechanism uses multiple parameters, such as load transmission and connection delay, to estimate nodes' excessive load. The proposed approach supports dynamic and interactive events. Even though this mechanism for load balancing has a good scalability feature compared to other methods, its performance in Petascale systems has multiple challenges. In Alowayyed et al. (2017), multi-dimensional computing algorithms are proposed for load-balancing functionality. Besides, these algorithms provide fault tolerance and energy management in distributed computing systems to redistribute the load. This paper presents three multi-dimensional models: Extreme Scaling, Heterogeneous Multiscale Computing, and Replica Computing. In this paper, the manner of multi-axis Constance, Falkon is a centralized task scheduler that uses simple sequential scheduling for Multi-Task Computing (MTC) (Wang et al. 2014).

In Bakhishoff et al. (2020), the authors present a mathematical model for managing dynamic and interactive events affecting the operation of the distributed load balancer based on the Discrete-Time Hidden Markov Model. The mathematical model presented in this article provides such a capability for the load balancer that, after analyzing each dynamic and interactive event, manages the activity based on the changes that cause the functionality of the distributed load balancer to be violated. In this article, the concept of the system state is considered to describe the function of the distributed load balancer and the return of the system's state to a stable condition after the dynamic and interactive event occurs as a method to manage the effects of the dynamic and interactive event on the functionality of the distributed load balancer. The most critical challenge governing the aforementioned mathematical model is in creating the state of the system corresponding to the functionality of the distributed load balancer. If the distributed load balancer uses more variables than the traditional one to describe its state, creating the system's state becomes complicated. Another challenge of this solution is the inability to change the system's state to a stable condition concerning the functionality of the distributed load balancer.

In Wylie (2020), the authors investigate the issue that if the processes are not correctly allocated to the resources, especially the computing resources defined in the Distributed Exascle systems, there is a possibility of the failure of the activities related to the distributed load balancer. In this article, in addition to this challenge of not correctly assigning processes to computing resources, other influential factors for breaking down the function of the distributed load balancer have been investigated. This solution, using the dynamic distributed load balancer (HemeLB) based on process analysis and feature extraction, has managed the effects of dynamic and interactive events on the functionality of the distributed load balancer.

In Lehman et al. (2019), based on the XQueue concept, a dynamic and interactive load-balancing mechanism is considered. This load-balancing mechanism uses a developed model based on the XQueue concept because it must consider dynamic and interactive events and their effects on the operation of the distributed load balancer. The effort of the article is that the proposed solution should be in such a way that the functionality of the distributed load balancer, considering the development of XQueue, is equivalent to the functionality of the distributed load balancer based on XQueue without considering dynamic and interactive events.

The centrality of the work is based on the fact that the load balancer analyzes the state of the computing elements based on an index which is conventionally the state of the CPU usage. This indicator can take more complex forms. The load balancer redistributes the load based on skill processing mechanisms if the state of the calculation element description index changes beyond a specific limit. In these mechanisms, a dynamic and interactive concept is not usually considered.

In the mechanisms that consider dynamic and interactive events, the design of the load balancer is based on considering the unique situation. In these mechanisms, an attempt is made to analyze the impacts of the dynamic and interactive events on the functionality of the distributed load balancer in one or more specific and determined situations. The mentioned conditions describe the constraints and limitations governing the load balancer function in a specific and confident way. A specific and detailed description of the function of the load balancer makes the situations that

can be described for the function of the load balancer after the dynamic and interactive event occurs. If, at the time of the dynamic and interactive event, the load balancer is in a state other than the one described in the proposed solution, then the proposed solution cannot respond and manage the impacts of the dynamic and interactive event on the functionality of the distributed load balancer. This event is because the functional nature of the load balancer is based on the implementation of logical sets of activities to implement activities related to the load balancer. In this article, we want to focus on the management of a load balancer in its specific and defined functional state, the activities and functions of the load balancer to implement the load balancer activities, and how to manage the effects of dynamic and interactive events on each of the activities constituting the function. This event makes it possible to provide a general solution for managing the effects of dynamic and interactive events on the functionality of the distributed load balancer.

3 Basic concepts

3.1 Load balancing definition

In traditional computing systems, from a load-balancing point of view, what describes the status of the process is time. The load-balancing element could consider the process three times <execution time, total time, idle time>. Execution time signifies the time allocated for a procedure. Full-time shows the time the process needs to complete its tasks. Idle time indicates when the process is not controlling the central processor. Defining the related elements to load balancing was also created based on the time-based management mentioned. Therefore, determining the processing element from a load-balancing point of view is based on the time it can provide processing power for computing processes. Thus, in computing systems, the adaptability of process requests over resource characteristics is defined by load balancing as a mapping between resource status descriptor and process variables, as shown in Eq. 1.

$$Load\ Distribution : \overbrace{[Process_{requirment}]}^{dependence\ space} \overbrace{\rightarrow}^{mapping} \overbrace{[Resource_{Space}]}^{independence\ space} Therefore$$

$$Best_{Load-Distribution} = \left[\left[\left(\nexists Process\right) \in HPC_{Process} \overbrace{\therefore}^{So\ Means} \ni \left(HPC_{Process-Scheduling}\right)\right] and \left[Resource_{Activity} Equal\ or\ near\ 100\%\right]\right] \quad (1)$$

As shown in Eq. 1, the functionality of the load balancer is a mapping of the process required for accessing

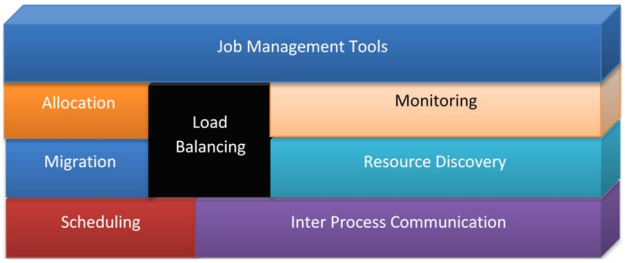

Fig. 1 Framework of load balancer with other constituent elements of the computing system manager

the central process to compute resource space in the system (Khaneghah et al. 2018). This mapping should allow the computing process to gain resources quickly and complete its task. Secondly, resource allocation between processes by load balancing should be done in a way that takes advantage of the maximum computing resource capabilities of the system. The definition of load distribution in Eq. 1 shows that it has based on two spaces process computing requests and computing resources capabilities. If only time is considered as the descriptor of resource and process elements in the simplest form, then the mapping in Eq. 1 forms Eq. 2.

$$Load\ Distribution : \overbrace{Time_{Request_{Process}}}^{main\ depended\ variable} \in \overbrace{Available\ Time_{Resource}}^{basic\ space\ for\ HPC} \quad (2)$$

The second part of Eq. 1 is also the actual for this equation. In Eq. 2, the definition of load distribution is based on the concept of time. Suppose, at any point, load distribution is activated, and there exists a process that requires a time limit for accessing the central processor. In that case, the load distribution element allocates the resources to the computing process based on the idle time of computing resources. As shown in Eq. 2, in this condition, the load distribution element should be able to allocate resources to processes so that the time requirements of the process are met by the idle time of computing resources. Both constituent variables in Eq. 2 are scalar.

Load distribution elements can include other characteristics for describing processes and computing resources. If the load distribution considers n dimensions for the description, then Eq. 1 turns into a mapping of $n \times n$. Determining the number of dimensions in Eq. 1 depends on how the space of the computing process and its requests is defined. For each computing system, a need exists to represent the computing process from the load distribution point of view.

The relation of load distribution in computing systems to other constituent elements of system management is defined based on the framework, as shown in Fig. 1.

As shown in Fig. 1, load balancing is the main element in the management of computing systems. The nature of the system management element is such that load balancing is responsible for the responding structure of the system. The load distribution element should do its responsibilities perfectly in any computing system based on the system's condition so that the system holds its maximum performance. Since load balancing connects to processes and computing resources of the system, it has precise information about the current condition. This event holds for both centralized and also decentralized distributed computing systems. In distributed systems, since load balancing is connected to the scheduling element, it has complete information about the local resources of the computing system. Also, because of load balancing's connection to Inter Processing Communication (IPC), the possibility of interaction with other computing resources in the system is defined. Load balancing uses migration and resource discovery elements as management tools. It uses the migration element for process migration and resource discovery to find system resources. The monitoring element, as the tool for monitoring the status of processes and resources of the system, and the allocation element, as the resource allocator for processes, have direct connections to the load balancer. Job management tools are also defined at the highest level in the framework presented in Fig. 1. The architecture of the load balancer shown in Fig. 1 is based on the architecture illustrated in Fig. 2.

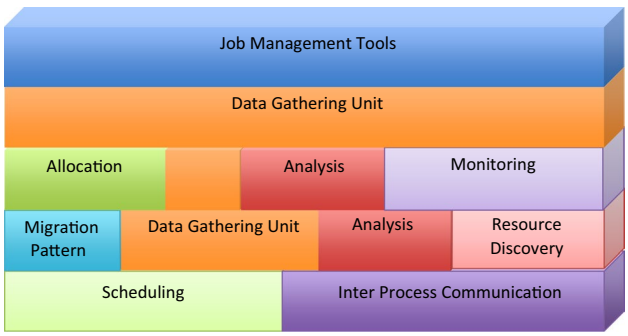

Fig. 2 Constituent elements of load balancer in the system manager framework

In Fig. 2, load balancing has three tasks: collecting data about the resources and processes, analyzing that information, and deciding on process transmission in computing systems (Khaneghah et al. 2018; Mirtaheri and Grandinetti 2017). This element uses existing databases in scheduling, IPC, and allocation units for data collection. On the other hand, load balancing uses databases for data gathering, resource discovery, and monitoring units to analyze the information. The migration management unit does the task of process migration.

In traditional computing systems, load balancing does not gain information about the request nature during the execution of the scientific application. It only collects data about the metrics of resource and process request status. This event is due to the nature of the programs executed in traditional computing systems. The computing system designer views the requests' character and the computing processes' requirements. Therefore, this makes it possible to select the appropriate mechanism for the load balancer by looking at the nature of the requests and the conditions that may occur during the execution of the scientific application. This mechanism should be able to describe the status of the process based on indicators to make decisions about the requirements of the process. The mechanism should also have indicators to tell the status of the resource, based on which it can decide whether the resource is capable of responding to a process request. In traditional computing systems, the patterns of request process events are a definite set. The computing system designer can describe a specific collection at the time of system design, called the Computing Process Request Pattern Set (CPRPS). The mechanism of the load balancer is determined on the basis that it can respond to this set during the execution of the scientific program. At the time of determining the mechanism of the load balancer, the following should be specified: (a) pattern of process and resource status, (b) mechanism of how to collect information, (c) determining which event patterns in the process status description indicate one of the events in existing patterns of CPRPS, (d) what pattern in the source status description state can respond to the pattern formed in CPRPS.

In traditional computing systems, the CPRPS sets and the response pattern to process requests are specified for the system designer based on the Resource State for Process Request Set (RSPRS). Therefore, the tasks of the load balancer consist of (a) collecting process status description information, (b) adapting the process status description to the CPRPS set, (c) determining pattern occurrence, (d) collecting source status information, (e) matching source status information with RSPRS and finally (f) transmission of the process using the process migration management

unit of the source processing element to the processing element that can respond to the process request.

3.2 Request B is a traditional request

The request is first sent to the local operating system in this case. The local operating system cannot respond to request B, so B is sent to the computing system's management unit. The management unit sends the process request to the load balancer. The load balancing creates a CPRPS set for request B. After the formation of the CPRPS set, the load balancing according to its implementation pattern (centralized, semi-centralized, or distributed), as well as the mechanisms used to collect data, decision making, and process migration, tries to create its own RSPRS set. The data collection mechanism of the load balancer can make its RSPRS set either after the CPRPS collection, at specific intervals, or before the formation of the CPRPS set. The decision-making mechanism for the load balancing seeks to find a processing element at the system level that can respond to request B by adapting the CPRPS set to the RSPRS. Based on its process migration mechanism, the load balancer transmits the process responsible for request B to the destination processing element. Traditional Cluster and Grade computing systems differ in the RSPRS set's pattern. The RSPRS set is formed as shown in Formula 3.

$$RSPRS \overbrace{::}^{define} \{\{R_1, [A_{1,1}, \ldots A_{1,i}, \ldots, A_{1,n}]\}, \ldots, \{R_m, [A_{m,1}, \ldots, A_{m,j}, \ldots A_{m,n}]\}\} \tag{3}$$

As shown in Formula 3, the set, RSPRS, is a set of m elements in which each element is in the form $< Resource, ResourceAttributevector >$. In this set, R_m represents the m^{th} source, which the load balancer collects its data. For each element R_m A vector containing n attributes is considered (the number n can be a different value for each source). In this vector, $A_{m,j}$ represents the status, or value of the descriptor of the j^{th} attribute of the R_m source. In Formula 3, $A_{m,j}$ can be a numerical or descriptive value. In traditional cluster computing systems, the number of elements that make up the RSPRS set is a numerical constant. In such systems, for each source R_m, the number of attributes describing the state of the source R_m is a fixed number, and only during the program's execution do the values of the vector describing the state of the source change. The number of elements that make up the RSPRS suite varies in peer-to-peer computing systems. In these types of systems, the number of vector properties for description associated with each source R_m does not change. The lack of change in the number of properties related to the descriptor vector of each source R_m is due to the precise control and management structure related to responding to request B.

Suppose the constituent elements of the RSPRS set are specific. The computing system is a traditional cluster system; when designing the computing system, all the elements of the RSPRS suite are known. Another condition is that the system design provides all RSPRS members' suites to the system management unit. If the designer cannot decide what elements are part of the RSPRS suite during the global activity process, then the computing system is DECS. Still, the designer at the time could not provide all members of the RSPRS set to the system management unit, even though the mechanism and pattern of finding these resources by the computing system management were precise. In that case, this computing system is a peer-to-peer computing system.

Traditional computing systems used a single pattern for the CPRPS set. In Eq. 4, the general form of the CPRPS set is introduced:

$$CPRPS \overbrace{::}^{define} \{Request_{type}, Request_{domain}, Request_{dependency}, \text{Re}quest_{influence}, [R_1, \ldots., R_z]\} \tag{4}$$

in which we have:

1. The $Request_{type}$ variable indicates the type of request. The computing system management unit can define acceptable values for the $Request_{type}$ variable based on the resource definition pattern, the process requirement pattern, and the nature of the program (or programs) running on the computing system. In traditional computing systems, the acceptable values for the $Request_{type}$ variables are specified at design time and do not change during the execution of the scientific program in the system; for example, processes always request access to the CPU source. This state causes the variable $Request_{type}$ to accept only the CPU value. The $Request_{type}$ a variable can not be limited to the resource type, and the system management unit can define any other value for it (Khaneghah 2017; Bakhishoff et al. 2020). The value defined for the $Request_{type}$ a variable must have a structure that responds to requests of the type specified by the $Request_{type}$ variable. One of the most critical features of the $Request_{type}$ a variable is an ability to define it at design time or the ability to define it at execution time. Using either of these methods constitutes the use of pre-embedded response structures or runtime response structures.
2. The $Request_{domain}$ variable indicates in which domain the system management unit should answer the pro-

cess request. Acceptable values for this variable, such as the $Request_{type}$ the system management unit specifies a variable. Typically, this variable can include local, Region, System, Global, or Representation (Bakhishoff et al. 2020). Based on the value of this variable, the load balancer in traditional computing systems decides what processing elements the RSPRS should contain. Acceptable values for the $Request_{domain}$ variable, in traditional computing systems, are specified by the designer at the design time. This variable's value affects the system's structure, whether Centralized, Decentralized or Distributed. The variable's value always equals the system in traditional Cluster computing systems. In this computing system, the load balancer uses the information of all processing elements to calculate the RSPRS set. The load balancer in peer-to-peer Grid computing systems uses $Request_{domain}$ variable to decide on the maximum acceptable amount for the scalability of the computing system. Scalability of computing systems means changing the processing elements that can be members of the RSPRS set. Creating the RSPRS set means collecting information and specifying the time to manage the load distribution. Conversely, data collection using load balancing is time-consuming, and the computing system must execute more tasks for the load balancer rather than running the target application. On the other hand, choosing a value for $Request_{domain}$ variable leads to fewer members in the RSPRS set. The load balancer has to create an appropriate proportion between the two concepts of time that is acceptable for the execution of the load balancer and acceptable values for $Request_{domain}$ variable. Without going into the details of resource discovery in peer-to-peer computing systems, the concept of scalability in these systems is to add processing elements (or elements) to the RSPRS suite that leads to a response to request B.

3. $Request_{influence}$ and $Request_{dependency}$ variables: When in a computing system, request B is generated by the computing process, it may cause changes in the units, or concepts of the computing system, in global activities, and especially in the local computing system. In the process of creating request B, elements and concepts may influence the formation of the request. In traditional cluster computing systems, the susceptibility of the request from the system is at the lowest level, and even it can be said that in this type of computing system, the effect of request B from the system is close to zero. In this type of computing system, when request B is generated in a process, the request is necessary for the CPU type, so the CPU queue of the local processing element is affected by this request. The nature of request B may be such that the processes connected to the process with request B need to be synchronized, which affects the connected processes. The load balancer must change the RSPRS and CPRPS sets under the influence of this process if the time requirements of the requested B are not met in the local processing element and consequently has to use the process migration unit. In peer-to-peer computing systems, the effect occurs in traditional cluster computing systems when request B is generated in a process. The most important effect of request B from the computing system is the ability to perform more than one global activity in the computing system. In this case, request B may be affected by requests from other global activity processes running concurrently on the local processing element. This influence can be due to requesting access to a shared resource in the local processing element. What makes the concept of the influence and susceptibility of request B on the elements and concepts of the computing system to be wholly controlled in traditional systems is the existence of management and control structures created at the time of system design for responding to request B.
4. The matrix $[R_1, \ldots, R_z]$ is a matrix $Z \times 1$, each of which is a part of the request B. Dividing request B into Z sub-requests, using a load balancer at the computing or operating system level, allows each processing element of the system to execute part (or parts) of the request separately. There are different patterns for dividing each request B into Z sub-requests. In Bakhishoff et al. (2020), as a peer-to-peer computing system, it uses the operating system model to separate request B into four sub-requests. Each request can generally contain one (or more than one) of the four I / O, Memory, File, or Process requests at the operating system level. In traditional computing systems, typically, each request B only includes a request to access the CPU or process. Therefore, request B can be described by a 1×1 matrix in traditional computing systems.

3.3 Dynamic and interactive events

Due to its direct relationship with the process and the resource, the load balancer is entirely dependent on the concept of system complexity and the complexity of scientific applications in need of high performance computing systems. The distributed system manager needs to use more complex patterns if the scientific program becomes complex or the state of the system elements (source or process) changes. The load balancer must use information (both in its simple form and complex form) to create an optimal match between the processes' requests and the resources' characteristics, to maintain the system's structure to continue operating.

In complex computing systems, events may change the status of processes and computing resources in the system.

In the management and control of the mentioned events, the load balancing should be in such a way that at any moment when the load balancing is activated, it must be able to maintain the structure of the system to continue operating. Load balancing matches the processing requests and resources in the system. Changing the status of processes and computing resources in complex computing systems affects the concepts, functionality, and relationship of the load balancer with the other system management components. DECS is one of the implementations of complex computing systems.

DECS are computational systems designed to execute applications that are dynamic and interactive. In this type of computing system, both processing elements and computing resources can be changed during program execution. The nature of computational processes is such that during the execution of the computational process, an event may occur in the system that leads to the definition and creation of a new requirement in the system. Such a requirement is not taken into account while designing the system.

In this computing system, the purpose of executing scientific applications is to reduce response time and use processing power to discover the laws governing natural events. In DECS, the program is a set of basic rules governing a natural event and uses these rules to discover other (or primary) laws governing a natural event.

This state allows processes running on a DECS to execute events that create dynamic and interactive events. In Innocenti et al. (2017), it is stated that dynamic and interactive events can be due to new request formation of new interactions (and communications) within or outside the system. These events cause the system to face a request it cannot respond it. This requirement was not considered in the initial responding structure for executing the application on this system.

To investigate the effect of dynamic and interactive events on the function of the load balancer, assume that the peer-to-peer distributed computing system in Bakhishoff et al. (2020) is running a scientific application of a dynamic and interactive nature (http://www.deep-project.EU). At the moment $t = \alpha$, in the ω processing element, request B is generated. Depending on the nature of the request B, one of the following two scenarios occurs. Request B can be either a request with a temporal nature or a request for access to the resource (either a processing resource or any other resource defined in system management).

4 Request B of dynamic and interactive nature

In DECS, in addition to changing the number of constituent elements of the RSPRS set, each source R_m, the number of constituent properties of the description vector of the source R_m It also varies during system runtime. Since this request is of dynamic and interactive nature, the load balancer encounters a request that nature is unknown to the unit. The unclear nature of the request may cause the load balancer to need to receive information other than the set of information collected about a particular source to respond to it. Therefore, in Exascale distributed computing systems, both the number of constituent elements of the RSPRS set and the number of constituent properties of the vector describe the state of each R_m source change over time.

In distributed Exascale systems, not all of the $Request_{type}$ variable's values can be specified at design time, and acceptable values for it change during the execution of the scientific application. Also, in this type of computing system, three elements of the system designer, management unit, as well as processes with dynamic and interactive nature can define the value for the $Request_{type}$ variable.

To have explicit knowledge of the acceptable values for the $Request_{type}$ the variable allows the load balancer to decide on the concept of computing system zoning and also on which element (or elements) has a higher ability to respond to requests with a specific value in $Request_{type}$ variable. If the acceptable values for the $Request_{type}$ a variable is known, and if an acceptable amount of time of the process's life has elapsed, the load balancer can decide on the continuation of execution in the process as soon as a request occurs by observing the value for the $Request_{type}$ variable. In DECS, the possibility of defining an acceptable new value for the $Request_{type}$ variable by the system management unit, as well as processes of interactive and dynamic nature, leads to the creation of new control and management structures for continuing the execution of applications. Creating new management and control structures based on the $Request_{type}$ variable causes the load balancer to change the RSPRS set.

In Exascale computing systems, the $Request_{domain}$ a variable can take on new values due to the possibility of defining a new global activity. This state expands the concept of the computing system. In peer-to-peer computing systems, the primary purpose of scalability is to obtain new computing resources to continue implementing activities related to scientific applications. While in DECS, the purpose of scalability, in addition to the mentioned concept, is the need for the load balancer to create new control and management structures to respond to dynamic and interactive requests. In DECS, the $Request_{domain}$ a variable must be able to consider these conditions as acceptable values. This event makes it possible for the load balancer to define the concept of Scalability + if the nature of the request is distributed Exascale systems are of dynamic and interactive nature. Scalability + is an expansion over the concept of scalability in which the goal of scalability. Scalability + primarily accesses new resources and creates a responsive structure to execute request B.

The two variables $Request_{influence}$ and $Request_{dependency}$ DECS assigns different values during runtime, depending on request B's dynamic and interactive nature. A request B with dynamic and interactive nature inevitably leads to the creation of a new process, either an administrative process or a process of managing new interaction and communication within or outside the system. This process is fully correlated with the process with request B, which causes the CPRPS set, $Request_{domain}$, $Request_{dependency}$, $Request_{influence}$ As well as $R_1, \ldots, R_z$ vector to change. In distributed Exascale systems, creating global activity responsive to request B is dynamic and interactive, creating a new set of influences and susceptibilities on other global activities. To analyze the effect and susceptibility of request B and the processes created to respond to it on other elements and concepts in the computing system (and especially other global activities,) each global activity can be described based on the Affine page so that the intersection of the pages with each other shows the influence and susceptibility of any global activity on other global activities (Mirtaheri et al. 2013).

In DECS, the members of the vector $R_1, \ldots, R_z$ may change during the procedure of responding to request B. Also, the number of constituent elements of the descriptive vector may change during the response to request B. This change is due to the interactions of the process that owns request B with the newly created process or other processes in the system and environment. Changing the components of the vector $R_1, \ldots, R_z$ means changing the requirements, which is the answer to request B. The load balancer should be able to take into account changes in the vector $R_1, \ldots, R_z$ during the response procedure.

5 ExaLB framwork

According to the discussion above, a framework similar to the framework shown in Fig. 3 can be considered a framework for the definition and function of the load balancer in distributed Exascale systems.

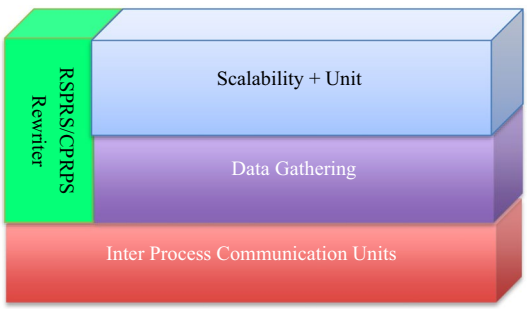

Fig. 3 Framework for defining the load balancer in DECS

As seen in Fig. 3, the load balancer requires the consideration of units that can address the decisions over the nature of the request. The load balancer needs units such as creating appropriate response structures to continue the global execution, considering the effects of executing more than one global activity, and the concept of the system environment.

6 RSPRS/CPRPS rewriting

As stated in Sect. 3.2, the request B type effectively forms the RSPRS set. The analyzer section of the load balancer makes decisions about the condition of the RSPRS set based on the following model, given the nature of request B.

1. If B is of the traditional type and responding to it does not change the number of resources in the system. In this situation, the number of resources that make up the system is constant. The characteristics that describe the status of resources are also fixed. The fact that the above two values are constant causes the two variables m and n to have fixed and definite values in Formula 3. This state allows the existing RSPRS set to be used to respond to request B without the need for any modifications.
2. Request B is of the traditional type, but it is impossible to respond to it using the resources available in the system. In this situation, by calling the resource discovery, the load balancer first expands the system intending to find a suitable resource to respond to the process request. Then, by examining the new set of RSPRS, which has been obtained from the addition of the mentioned resource to the previous set, it maps the two sets of the request pattern and the response pattern to each other and tries to establish the balance of load in the system. In this case, in formula 3, the variable m has a non-fixed value, and the variable n has a fixed value.
3. Request B is dynamic and interactive. In Formula 3, neither of the variables m and n have a constant value due to the system's need for scalability and the lack of information about the cause and nature of request B, respectively. In this condition, the load balancer must first identify the nature of the request and then explore the characteristics of the resources with which it is possible to respond. In this case, the RSPRS set can respond to request B after making fundamental changes.

The task of the Nature Request unit is to determine the nature of the request. This event causes the need to define the request Nature unit in the load balancer in DECS. This unit must determine whether the request formed is of type (a) or (b). This element can decide whether the nature of the request is traditional or due to the dynamic and interactive occurrences based on the processor's behavior, the

history of the activities, and the concept of annihilating polynomials (Wylie 2020). Due to the multi-dimensional nature of the process requests and the type of resources required to respond to requests in DECS, Formula 1 can be rewritten into Formula 5 based on the vector pattern.

$$F(Load\ Balancing):\ \overbrace{[Req_1 \dots, Req_N]}^{independence\ variable\ define} \rightarrow \overbrace{[Resource_1 \dots, Resource_M]}^{dependence\ variable\ define} \overbrace{|}^{it\ means\ that}$$
$$\left[\forall i \in (\{1,\dots,N\}, Req_i) \overbrace{\therefore}^{in\ such\ a\ way\ that} [Req_{type} Req_{Behavior} Req_{history} Req_{NULL}]\right] and \tag{5}$$
$$\left[\forall j \in \{1,\dots,M\}, Resource_j \overbrace{\therefore}^{in\ such\ a\ way\ that} [Resource_{type} Resource_{fuctionality} Resource_{history} Resource_{NULL}]\right]$$

Formula 5 is the vector form of the load balancer's functionality. According to Formula 5:

Each process can be described as a $1 \times N$ vector in which N is the number of process requests executed in each processing element ω.
Each source in the processing element ω can is described based on a $1 \times M$ vector in which M is the number of responses provided by the computing resource to the global computing processes.
Each process request is executed in the processing element ω based on the 2×2 matrix in the form $[Req_{type} Req_{Behavior} Req_{history} Req_{NULL}]$.

In the 2×2 matrix mentioned, Req_{type} Indicates the type of request, which tells what source the process request is available. The load balancer must be able to classify the resources in the computing system to extract the Req_{type} data. In the case of traditional computing systems, such as cluster, Grid, and peer-to-peer computing systems, Req_{type} In Exascale peer-to-peer computing system (Khaneghah 2017), each process request can be for one of four resource types of file, input/output, memory, or processing element is always of CPU type. In DECS, the request type can be sources other than the process resource.

Listed in the 2×2 matrix mentioned above, $Req_{history}$ indicates the history of the request. From the point of view of the load balancer, each request for a computational process is either an independent request created for the first time or a follow-up request that has resulted from responding to another request. Maintaining a request history allows the load balancer to extract information about the resources required by the process and processing elements connected to the processing element ω and respond to the request. From the point of view of the load balancer, the rule stated in formula six is held for each element of the data structure of the link list in $Req_{history}$.

$$\forall Req_{history} \exists! \lambda : \lambda \in Resource_{history_{omega}} \tag{6}$$

According to Formula 6, for each element defined in the $Req_{history}$ a data structure, there must be a corresponding member in the $Resource_{history_{omega}}$ a data structure that represents the source by which the request was answered. This resource can be one of the local resources of the omega computing system or a source outside the omega computing element. Considering Formula 6, the $Req_{history}$ data becomes a linked list structure in which two pointers are defined. The first pointer refers to the subsequent subordinate request resulting from the response to the request, and the second pointer to a corresponding element in the data structure, $Resource_{history_{omega}}$.

In the 2×2 matrix mentioned, $Req_{behavior}$ represents the request behavior. The load balancer can consider different states for $Req_{behavior}$. The primary purpose of this element is to investigate whether the behavior of global activity's processes follows a particular pattern. This knowledge is implemented as a link list in which each node represents the unique properties of a process. From the point of view of the load balancer, a process can have a unique attribute (or attributes) regarding the three concepts of time, location, and dependency. These unique attributes of the request process define the request's behavior. The behavior of each process request that constitutes the global activity follows formula 7.

$$f(Req_{process}) \overbrace{\alpha}^{defined} \left([Time \in [A,B]] \overbrace{\blacksquare}^{undefinedoperator} \sum Location \blacksquare \sum Dependency \right) \tag{7}$$

In Formula 7, the request behavior of a global activity process consists of three concepts: the time limitation (or limitations) governing the request, the location limitation (or limitations) governing the request, and the dependence of the request on any other concept. The two limitations of location and dependency are discrete. Time

limitations are of continuous interval type. In the case of a global activity constituent process, some of these attributes may have NULL values. The value of NULL means no limitation on that particular feature. The relationship and impact of these attributes on each other are determined for each process request. In the case of a particular request, certain limitations may be more critical and affect the value of other limitations. Ideally, the result of these three features on each other indicates the behavior of the process request.

Conventionally, the T, L, and D limitations must be independent. $Req_{behavior}$ is maintained as a linked list in which each element is of the form $< T, L, D >$. In the procedure of global computing process execution, if during each execution of the process, one of the three elements of the $Req_{behavior}$ changes, one element is added to the linked list. The load balancer has access to the data set related to the three variables $< T, L, D >$ at each process execution time. It can use the expected value of this data to decide on the independence or interdependence of one another. If Formula 8 is valid for each execution of the process, then two of the variables in $< T, L, D >$ are of the independent type, and otherwise, they are interdependent.

$$\left[\forall i,j \in \{D,L,T\}\right] :: \left[E(i+j) = E(i) + E(j)\right] i,\ j\ is\ independence \tag{8}$$

In Formula 8, two variables' dependence and independence, are a time function. This event causes a) the load balancer to use regression if not independent for calculating the operator between two variables, and B) two variables can be independent of each other for a particular time and then interdependent, or vice versa.

In the 2×2 matrix mentioned in formula 5, Req_{NULL} is the main request and the reason for the formation of global activity. Each global activity is formed in a computational element, such as ω, to respond to a specific request. This specific request can be described in the form shown in Formula 9.

$$Req_{Null} = \left(A_i x_i + B_j y_j\right) \tag{9}$$

Formula 9 states that each element of $Resource_{NULL}$ Is a linear polynomial in which x_i represents the resources required to meet the main request of the global activity, A_i represents the weight and importance of the resources requested, y_j represents the limitations, Application limitations, and B_j indicates the weight and importance of the limitations. Formula 9 can be rewritten for each B request.

$$\begin{aligned} Req_{NULL} + Req_{\beta}(T) &= \left(Req_{NULL}(T) + Req_{\beta}(T) and Req_{NULL} * Req_{\beta}(T)\right) \\ &= \left(Req_{NULL}(T) * Req_{\beta}(T) and Req_{NULL}(T)\right) = 0 \end{aligned} \tag{10}$$

In Formula 10, T is a linear conversion on the request space. This linear conversion represents responding to the request, which is answered under the linear T conversion. If Formula 10 is true, request B is a formal request; otherwise, request B is of dynamic and interactive nature.

4. Each response provided by the source in the processing element ω is defined based on a 2×2 matrix in the form of $\left[Resource_{type} Resource_{fuctionality} Resource_{history} Resource_{NULL}\right]$.

In the 2×2 matrix mentioned, the $Resource_{type}$ element refers to the responding source to the request. The computing system manager can use any pattern to classify the resources in the computing system. This classification must match the Req_{type} classification defined in the request matrix but does not have to be done as one by one adaption. The manager of the computing system, based on any function (or even, in some cases, based on any relations), can establish a match between the two mentioned concepts. In the computing system manager of Khaneghah (2017), a one-to-one match between the process request for resources and the manageable resources by the system manager is used based on the operating system's pattern for the classification of resources. The load balancer uses $Resource_{type}$ Information to define the concept of resource attributes. In this situation, the load balancer defines a set of indicators for each type of resource defined in the system manager unit. Any data structure at the operating system's kernel can maintain information about these indicators.

One of the most important differences between various load balancer implementations is determining indicators of resource descriptors. The higher the number of indicators the load balancer uses, the more accurate it is in resource status. A clear view enables the load balancer to create an optimal match between the process request and the responding resource. On the other hand, using more indicators to describe the resource increase the time required to gather information about them. In traditional computing systems, the $Resource_{type}$ In DECS, there may be a process request that (A) does not relate to the type of CPU resource and (B) can not be analyzed by the conventional attributes of resources that the load balancer uses to describe the resource. Typically includes the CPU computing resource. Indicators describing the CPU resource are idle and busy times, or the resource is either available or unavailable.

In distributed Exascale systems, the load balancer typically uses the concept of response history to examine the number of indicators and whether the indicators can respond to process requests. For this purpose, the load balancer utilizes a function as one in Eq. 11.

$$P(t, req) = \left(\frac{Acceptable - Acceptable_{global}}{\#_{Req}(t)} \right) \quad (11)$$

As can be seen in Formula 11, the load balancer in distributed Exascale systems is a binomial function to check whether the defined indicators are capable of responding to process requests. This function is a two-variable function of time and request, independent variables. The mechanism of function 11 is based on the fact that in the case of a specific request Req_A Examines how successfully the request has been answered locally. The local response level means that the process is initially created locally or is part of a global activity intended to run locally in the processing element. If this amount is more than a certain number, the defined indicators are appropriate, and the load balancer does not need to change the defined indicators. If less, it means there is a need for defining new indicators for the resources in the computing system.

In Formula 11, according to the model presented in Khaneghah (2017), since only the number of global processes is known, to calculate the number of locally responded processes, the difference between the number of global processes responses and the number of all processes responses is used.

The $Resource_{history}$ indicates the history of resource usage. The load balancer for each resource defined in the computing system maintains two types of source information related to the resource (with its descriptive indicators) and historical information related to the responses of the resource to requests. The second type leads to the creation of a shared data structure between the $Resource_{history}$ and $Req_{history}$. Historical information about descriptive indicators represents the model the load balancer uses to describe the resource for the processes in the system. The more dynamic and interactive the processes in the system, the more varied the patterns used by the load balancer to describe the resource.

$Resource_{functionality}$ indicates the functionality of the resource. This knowledge can be described based on variables such as time, cost, and type of response to existing requests. In traditional computing systems, the resource's function is considered time-independent. In such systems, the load balancer assumes that (A) the function of the resource α is fixed during the execution of the scientific application, and (B) the independent variable describing the resource α is constant and equal to the measurable indicator of the resource's operating time. In DECS, (A) the function of the resource α changes during the execution of the scientific application, and (B) more than one independent variable can be defined to describe the resource α, and, consequently, the measurable indicator of the resource. In this type of computing system, the finite vector of V_Alpha can be defined for the resource α, whose indices represent evaluation indicators or independent variables that describe the function of the resource. The load balancer uses a formula similar to Eq. 12 to determine what the function of the resource α is at the moment t, for the process Fi (which is itself part of a global activity).

$$f(T_{Alpha_{Fi}}(t)) = 0 \quad (12)$$

In Eq. 12, T is a linear operation on the finite vector space of V_Alpha. If f is equal to the polynomials defining the state of the resource functionality, f is necessarily a type of indicator describing the functionality of the resource Alpha if $f(T) = 0$. In this case, the function f represents the functionality of the resource Alpha at the moment t from the point of view of the resource management unit for the process Fi. In distributed Exascale systems, each resource Alpha can have different functions in terms of the elements that make up the system manager unit. In Eq. 12, the linear function T is the operator representing the function of the load balancer on the resource Alpha. This operator indicates what activities the load balancer considers to be definable and applicable to the resource Alpha. Based on Eq. 12, we can define an ordered base such as $\{V_{Alpha_1}, \ldots, V_{Alpha_m}\}$ for the V_Alpha vector, matrix U can be defined to represent T based on that. By considering the existing assumption, Equation No. 13 can be defined.

$$\sum_{j=1}^{m} \overbrace{(\sigma_{ji}T - A_{ji}I)}^{B_{ij}} \alpha_{j=0} | When\ m = 2\ then\ \ B = \begin{bmatrix} T - A_{11}I & -A_{21}I \\ -A_{12}I & T - A_{22}I \end{bmatrix} \quad (13)$$

Equation 12 is calculated based on formula 14 in the case where m is a finite number greater than 2.

$$det\ det(U)_t = f(T)_t \quad (14)$$

Equation 13 solves Eq. 12 in the condition that the resource Alpha is described based on two indicators in which their definition is not a function of a time-independent variable. Equation 14 solves Eq. 12 in the condition that the resource Alpha is based on the m indicators and is a function of the time-independent variable.

The matrix U is a description of the activity in time and is defined based on K, the displacement loop with the identical element consisting of all polynomials of T. In general, the definition of the U matrix is $U_{ij} = \sigma_{ij}T - A_{ij}I$. Equations 13 and 14 are solutions to find $f(T)$ at the definite moment t. In the most general case, to determine f (T), it is necessary to calculate the matrix U. The general concept defines the matrix U that any resource can be defined based on two sets of Z, which represents the resource type, and X, which represents the activities that can be performed on the resource. These two sets create K displacement rings with an identical element consisting of all definable polynomials of T. The

identical element consists of all definable polynomials of T, which embodies the concept of resource execution (serving) time. This concept can be defined as a familiar concept for all resources. Therefore, in Eqs. 13 and 14, an element I am equal to Busy Time. Two operations of and can are defined for two sets of Z and X. Action ⊗ is the possibility of acting on a specific resource type, and action ⊕ represents the element that performs the task. For example, in the case of the CPU resource type, the possible operations of allocation, reallocation, and the executing element for the scheduler.

In the 2×2 matrix mentioned, $Resource_{NULL}$ is the coefficient of resource importance in the execution of global activity. For each activity Fi, each resource Alpha has a coefficient that can be expressed according to Formula 15.

$$Resource_{NULL} = \left(A_i x_i + B_j y_j\right) \tag{15}$$

In Formula 15, each element of $Resource_{NULL}$ Is a linear polynomial in which x_i denotes global activities that use the resource Alpha, A_i denotes the time each global activity uses the resource Alpha, y_j denotes limitations over resources, and B_j indicates the weight and importance of the limitations.

Formula 15 can be rewritten to Formula 16 for each resource β.

$$\begin{aligned} &Resource_{NULL} + Resource_{\beta}(T) \\ &= \left(Resource_{NULL}(T) + Resource_{\beta}(T)\, and\, Resource_{NULL} * Resource_{\beta}(T)\right) \\ &= \left(Resource_{NULL}(T) * Resource_{\beta}(T)\, and\, Req_{NULL}(T)\right) = 0 \end{aligned} \tag{16}$$

In Formula 16, if the value of $Resource_{\beta}(T)$ $and Resource_{NULL} * Resource_{\beta}(T)$ is zero, then the resource β is a resource with a traditional pattern, and otherwise, the resource β is a resource capable of performing activities of a dynamic and interactive nature. Formula 16, T is a linear transformation of the resource β's space. This linear conversion indicates the use of the resource. The request is answered under the linear transformation of T.

6.1 Data gathering

The main task of the load balancer is to establish a proper allocation between the two sets of RSPRS and CPRPS. Ideally, this unit should have a clear view of both sets to carry out its tasks. In DECS, both the CPRPS and RSPRS sets can change at any point in the execution of the scientific application. If process Z in DECS has the request β at $t = V$, it causes the load balancer to be activated, in which case the load balancer creates an image of the system status at the moment $t = V$. Creating an image by the load balancer means creating the CPRPS and RSPRS sets associated with the request β and the process Z. Dynamic and interactive events may occur at any point in the system's execution. Consequently, CPRPS and RSPRS may change. However, the load balancer, from the moment $t = V$ to the moment of responding to the β request, considers the system status of the request β based on the dual pair RI:: <RSPRS, CPRPS>. Dynamic and interactive events can cause a significant difference between the real RI and the RI at the starting time of the distribution task. Sometimes, it violates the cause of the load balancer's activation (Khaneghah et al. 2018). For this purpose, RI must be converted from its traditional state to RI (t), which means that the two sets of CPRPS and RSPRS change from time-independent to time-dependent variables.

Rewriting RI based on the variable time means rewriting Eq. 5 with the independent time variable in Eq. 17.

$$\begin{aligned} &RI(t) :: \left[\forall i \in \{1, \ldots, N\}, Req_i \therefore \left[Req_{type} Req_{Behavior} Req_{history} Req_{NULL}\right]\right. \\ &\left.\left[\forall j \in \{1, \ldots, M\}, Resource_j \therefore \left[Resource_{type} Resource_{fuctionality} Resource_{history} Resource_{NULL}\right]\right]\right](t) \end{aligned}$$

$$\begin{aligned} RI(t) :: &\left(\frac{\partial_{\left[Req_{type} Req_{Behavior} Req_{history} Req_{NULL}\right]}}{\partial_t} \right. \\ &\left. \rightarrow \frac{\partial_{\left[Resource_{type} Resource_{fuctionality} Resource_{history} Resource_{NULL}\right]}}{\partial_t} \right) \end{aligned} \tag{17}$$

Equation 17 shows that taking a partial derivative of the $\left[Resource_{type} Resource_{fuctionality} Resource_{history} Resource_{NULL}\right]$ and $\left[Req_{type} Req_{Behavior} Req_{history} Req_{NULL}\right]$ matrices in terms of the independent variable of time allow each of the two sets of CPRPS and RSPR to be rewritten using the time variable. The independent axial variable of matrices $\left[Resource_{type} Resource_{fuctionality} Resource_{history} Resource_{NULL}\right]$, and $\left[Req_{type} Req_{Behavior} Req_{history} Req_{NULL}\right]$ are the resource and the request variables, respectively. According to the definition of axial variables and the need to rewrite based on the independent variable of time, Eq. 17 can be rewritten to Forms 18 and 19.

$$\begin{aligned} &J_F(resource, time) : \\ &= \frac{\partial\left(Resource_{type}, Resource_{history}, Resource_{fuctionality}, Resource_{NULL}\right)}{\partial(resource, time)} : \\ &= \left[\frac{\partial Resource_{type}}{\partial resource} \frac{\partial Resource_{type}}{\partial time} \frac{\partial Resource_{history}}{\partial resource} \frac{\partial Resource_{history}}{\partial time} \right. \\ &\quad \frac{\partial Resource_{fuctionality}}{\partial resource} \frac{\partial Resource_{NULL}}{\partial resource} \\ &\quad \left. \frac{\partial Resource_{fuctionality}}{\partial time} \frac{\partial Resource_{NULL}}{\partial time} \right] \end{aligned} \tag{18}$$

$$J_F(request, time) := \frac{\partial\left(Req_{type}, Req_{history}, Req_{Behavior}, Req_{NULL}\right)}{\partial(request, time)} :$$
$$= \begin{bmatrix} \frac{\partial Req_{type}}{\partial request} & \frac{\partial Req_{type}}{\partial time} & \frac{\partial Req_{history}}{\partial request} & \frac{\partial Req_{history}}{\partial time} & \frac{\partial Req_{Behavior}}{\partial request} \\ \frac{\partial Req_{NULL}}{\partial request} & \frac{\partial Req_{Behavior}}{\partial time} & \frac{\partial Req_{NULL}}{\partial time} \end{bmatrix} \tag{19}$$

Equations 18 and 19 are the rewrites of the matrices that make up Eq. 5, Regarding axial variables of resource and request and the independent variable of time. The most critical challenge in calculating Eqs. 18 and 19 is the time and number of times these Equations are to be calculated. The higher the calculation frequency of these Equations, the closer the calculated RI is to the actual RI. Ideally, for each unit of time, the load balancer performs Eqs. 18 and 19, where the calculated RI equals the actual RI. This event increases the cost of executing the load balancer, and this task would need a separate processing element so that calculations could be performed continuously. This event contradicts the nature of the DECS. Therefore, the load balancer uses the concept of a similar matrix in each processing element to calculate the RI close to the actual RI.

From the point of view of the load balancer, if two RI matrices related to the two permissible states are similar, it means that the dynamic and interactive nature of the computing system has not occurred. For this case, the load balancer uses the concept of the axial element. The axial element of the matrix described in Eq. 18 is the $\left[\frac{\partial Resource_{NULL}}{\partial resource} \frac{\partial Resource_{NULL}}{\partial time}\right]$ element, and in Eq. 19, is the $\left[\frac{\partial Req_{NULL}}{\partial request} \frac{\partial Req_{NULL}}{\partial time}\right]$ element. The load balancer decides on the similarity of the matrices by comparing the axial element of matrices at the moment $t = Alpha$ with that of the moment $t = Alpha + E$. The load balancer also determines the value of E during system execution. During this time (and calculating the RI), the load balancer can decide on the value of E for each activity, with which the matrices of Eqs. 18 and 19 differ from these matrices at the moment before E.

6.2 Scalability

One of the main features of distributed computing systems is the introduction to scalability. Resource discovery is the central concept of scalability in traditional computing systems (Lehman et al. 2019). The general mechanism governing the scalability in this type of computing system is based on the fact that a request occurs in the system that neither the local operating system of the process nor the load balancer can respond to. Therefore, the load balancer calls the resource discovery. Based on the type of resource requested by the process and resource discovery mechanism, the resource discovery tries to find a processing element outside the system and can respond to the process request. The mechanism used by the resource discovery indicates the structure of the response to the request and is specified when designing the computing system. Defining the resource discovery mechanism at the time of system design is possible due to having information about the response structure required for the implementation of the scientific application by the system designer.

The scalability mechanism in traditional computing systems increases the system's ability to respond to process requests with the arrival of computing resources and new computing processes. This event causes the load balancer to redistribute when scalability occurs. The load balancer must be able to extract the features and capabilities of each element added to the system (or delete information of each element removed from the system) to decide about the possibility of using that element (or the impossibility of using the removed element) in the procedure of responding to the process request(s).

In DECS, scalability has a different definition and functionality due to the system's possibility of dynamic and interactive events. The most crucial difference between scalability in traditional computing systems and DECS is that they have information about the nature of requests, the response structure, and the lack of events that could change the system's state or the environment during scalability.

In traditional distributed computing systems, the nature of the request does not affect the mechanism used to discover the resource or the scalability. The load balancer sends the triad of <Request type, Process, Time, and Location Limitation> to the resource discovery. The resource discovery creates a response structure based on its mechanism by considering the type of request and time and location limitations. In DECS, the nature of conventional requests differs from that of dynamic and interactive requests.

The nature of the request in this computing system refers to why the process has created a request that has led to the need for scalability. In conventional requests, the scalability response structure is created to respond to the process request based on the centrality of the resource discovery mechanism. On the other hand, when a request of dynamic and interactive nature occurs, the reason for the request is not apparent for the load balancing and, consequently, for the resource discovery. If the load balancer fails to determine the nature of the request, the resource discovery may create a response structure that cannot respond to the process request. Any situation, such as creating new processes or the need for interactions and communications within and outside the system, causes requests for a particular resource to be different from others (Khaneghah and Sharifi 2014).

The focus of resource discovery in this computing system is to find the processing element that, in addition to providing a resource that can meet the needs of the process, must also be in line with the nature of the request. In DECS, the load balancer sends a quadruple of < Request Nature, Request Type, Process, Time, and Location Limitation > to the resource discovery. Based on the pattern of arguments received from the load balancer, the resource discovery decides whether to use the traditional concept of resource discovery or the ExaScalability pattern for scalability. In DECS, the nature of the request determines which structure to be created for responding to the process request and which element with what features can respond to that request. In traditional computing systems, this is defined during system design while defining the mechanism for resource discovery.

The resource discovery in the computing system is affected by a set of practical factors. These factors also affect the response to the process request. In traditional computing systems, nothing happens in the computing system that affects the practical factors of the resource discovery during the resource discovery procedure.

It is assumed that in the distributed system of Khaneghah (2017), to understand the resource discovery scenario in a DECS, an event of dynamic and interactive nature occurs at the moment t = Alpha in one of the members of the computing system. Figure 4 shows the event status leading to scalability in the system.

As seen in Fig. 4, in machine O, a member of the computing system (marked with a lightning symbol), an event of a dynamic and interactive nature occurs. This event can be any of the situations described, the creation of a new process T, or the new interactions and connections within and outside a system. The computing system cannot respond to this request. Up to this point, the system status follows the pattern of a traditionally distributed computing system.

The difference between a traditional computing system and a DECS begins at the moment of request analysis by the system manager. In a traditional computing system, the manager recognizes the nature of the request, so it calls the resource discovery. The resource discovery discovers the resource that can respond to requests based on the predefined mechanism and policy. In DECS, when a request is generated in the system, the system manager does not know the nature of the request, so as shown in Fig. 4, it creates a new global activity for managing the request.

In Fig. 4, the occurrence of a dynamic and interactive event has caused the global activity in machine y to continue, and the manager of machine O has created a response structure for managing the request with dynamic and interactive nature. As with any global activity, the request can be answered on Machine X or transferred to other machines on the DECS.

Scalability in DECS, in addition to its traditional role, must have the load balancer to make decisions while calling for resource discovery or creating a new response structure. The new response structure should not violate system performance.

At the moment of load imbalance in DECS due to a dynamic and interactive event, the load balancer can take one of the two redistribution policies, either going to the following equilibrium or returning to the previous stable distribution status (Khaneghah and Sharifi 2014). In this case, we can define the Effort variable, which represents the coefficient of system call by the load balancer relative to the total system calls executed in a single unit of time. The $Effort_{Si}$ is calculated by Eq. 20.

$$Effort_{Si} = \left[\frac{\sum_{i=}^{MACHINEZ} (X_{LB})}{N_{SC}} \right]_{t=LB}^{t=LB+\epsilon} \tag{20}$$

As stated in Eq. 20, the $Effort_{Si}$ is equal to the number of system calls associated with the load balancer relative to the total system calls during the interval when the load

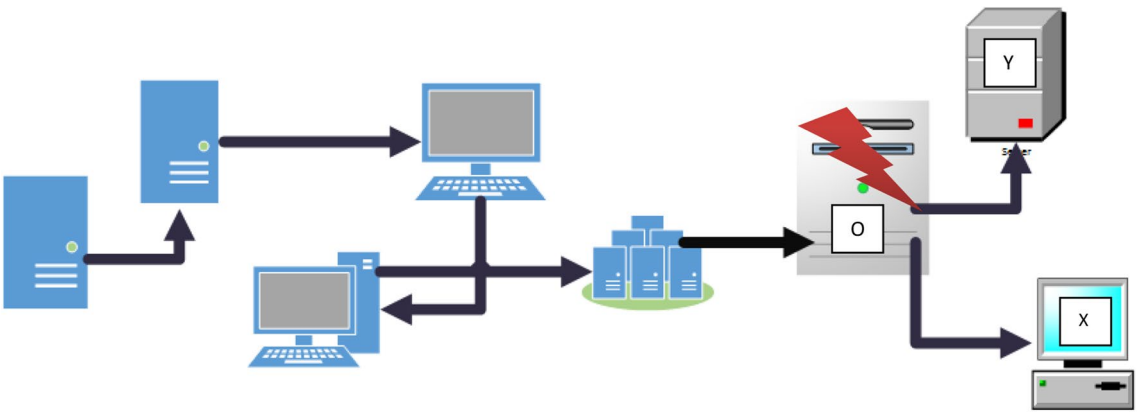

Fig. 4 Occurrence of the event leading to the activation of resource discovery in the computing system (Khaneghah 2017; Sharifi et al. 2010)

balancer is busy redistributing the load and creating a new state of balance in the system. In Eq. 20, the *MACHINEZ* is defined because there may be more than one beneficiary machine involved. The *MACHINEZ* variable is equal to the total number of beneficiary machines related to the load balancer's activities. In Eq. 20, the load balancer redistributes the load in the time interval [LB, LB + ε]. If the value of the *MACHINEZ* variable is more than one, the variable $Statistic_{Sij}$ As shown in Eq. 21, it should be used. The $Statistic_{Sij}$ can be calculated by Eq. 21. Note that the status of the load balance is denoted by i for the time before the load balancer is activated, and the status of the load balance after it is redistributed is shown by j.

$$Statistic_{Sij} = \sqrt{\left(\frac{effort_{si} * \left(1 - effort_{si}\right)}{N_{SC_i}}\right) + \left(\frac{effort_{sj} * \left(1 - effort_{sj}\right)}{N_{SC_j}}\right)} \tag{21}$$

In Eq. 21, the variable $effort_{si}$ represents the load balance of state i and the $effort_{sj}$ represents the load balance of state j. The reason for defining the variable $Statistic_{Sij}$ is that the two states i and j are independent of each other, and ϵ is acceptable in both $effort_{si}$ and $effort_{sj}$. If the value in either i and j is very large or very small, then that state cannot be considered a valid state, and that state can be used as the load balance of state i or j.

The standardized variable of $Standard_{Sij}$ can be calculated with Eq. 22.

$$Standard_{Sij} = \frac{\left[\left(effort_i^{Z=Server} - effort_j^{Z=Server}\right) - \left(effort_i - effort_j\right)\right]}{Statistic_{Sij}} \tag{22}$$

As can be seen in Eq. 22, to calculate the standardization function of *Statistics* for both states of i and j, the load balancer needs to have information about the status of system calls of the load balancer relative to the total system calls in the processing element in both states i and j. The load balancer using Eq. 23 can decide whether the distribution status is similar in both i and j.

$$P\left[\overbrace{\left[\left[effort_i^{Z=Server} - effort_j^{Z=Server}\right] - \left[Standard_{Sij}\right]_{\left(\frac{\alpha}{2}\right)} * Statistic_{Sij}\right]}^{Alpha}\right.$$
$$\leq \left(effort_i - effort_j\right) \tag{23}$$
$$\left. \leq \overbrace{\left[\left[effort_i^{Z=Server} - effort_j^{Z=Server}\right] + \left[Standard_{Sij}\right]_{\left(\frac{\alpha}{2}\right)} * Statistic_{Sij}\right]}^{Beta}\right]$$

The variable in Eq. 23 indicates the similarity coefficient of distribution in both states i, and j. The numerical value of α is between zero and 100. When in function number 23, α is equal to 100, then from the point of view of the load balancer, the two distribution states i and j are pretty similar in distribution. When the value of α is equal to zero, then status i and j are entirely different. If the value of $(effort_i - effort_j)$ is in the range [Alpha, Beta], then the value of α is correct; otherwise, the coefficient assigned to α is incorrect. The load-balancer can use Eq. 23 to decide on the system status similarity before and after adding a new resource.

Suppose, according to Eq. 23, the scalability reduces the system's performance. In that case, the load balancer should use an alternative scalability mechanism, such as a functionality change (Kayal 2009), or an extended version of the scalability, such as proxy openness (Khaneghah 2017).

7 Evaluation

The distributed peer-to-peer computing system of Khaneghah (2017) has been used to evaluate the ExaLB framework in DECS. In this system, the manager uses the concept of regions to manage the system. This state makes it possible to define four areas in this system that fit the four primary resources defined by the operating system. The concept of global activity is defined according to what is shown in Fig. 4.

Three types of global activities have been implemented in the system (Khaneghah 2017). Charm + +, MM5, and WRF software (Steen and Tanenbaum 2016; Adibi and Khaneghah 2020; Mirtaheri et al. 2013) are three software applications requiring high performance computing and extensive processing power. Each of this three software uses the computing resources in the system based on global activity. Therefore, three types of global activities are running in the system at the same time. Each system member can execute one, two, or all three activities at any time.

The considered computing system is a large-scale computing system in which the number of machines running scientific and practical programs is higher than traditional computing systems such as clusters. This computing system operates in a distributed manner. A part of this system has been used to evaluate the presented mathematical model. Therefore, the result obtained is due to the actual implementation of the program.

The computing system's number of elements equals 120 processing elements. Forming a system of 120 processing elements makes it possible to consider it an extensive testbed system for each of the three software. The applications mentioned usually run on a smaller number of processing elements, so their implementation of 120 elements makes it possible to analyze the status of the software when running in an extensive system.

Due to the nature of DECS, and the need to define the basic computing system, 40 processing elements have been considered for the direct computations. These 40 processing elements correspond to the basic requirements of processes related to the mentioned scientific application. During the execution, if processes request new resources to continue, the ExaLB load balancer uses the Scalability + unit to expand the system and add new resources.

To create dynamic and interactive events, machine no. 34 uses a particular version of system management software (Khaneghah 2017) in which the load balancer uses the ExaLB framework. The hardware configuration of this machine is compatible with other machines in the system. The reason for choosing processing element no. 34 is since this element participates in the global activities of all three programs, most of the time executing mentioned applications. If for each global activity, one activity page is considered according to Mirtaheri et al. (2013), in most of the applications' time, the processing element 34 is the intersection point of all three pages corresponding to the three global activities. Any other processing element could also be selected as the element under evaluation.

In element 34, the system manager unit has been changed. This state allows the unit in this element to manage the three situations of process creation, communication, and interaction with the environment, which leads to dynamic and interactive events. The manager unit of processing element no. Thirty-four can manage the relationship between the process related to the global activity and the corresponding process outside the computing system, which was not considered in the initial response structure of the global activity. Therefore, in processing element No. 34, the manager unit (A) can manage processes that are not present in the global activity structure at the time of system design, and (B) can manage the communications between two processes that make up the global activity, which was not intended for global activity, and (C) is in communication with another machine on which there are types of processes that make up the three scientific applications mentioned above.

Therefore, the manager unit of element no. Thirty-four can manage situations that lead to dynamic and interactive events in processes related to global activity. Figure 5 shows the occurrence of usual, dynamic, and interactive events and the number of events in each of the three situations that can be responded to by the ExaLB framework.

In Fig. 5, with regards to processing element no. 34, 53% of the requests leading to the activation of the load balancer are related to dynamic and interactive process creation requests, 19% are related to dynamic and interactive communication requests, and 6% are related to dynamic and interactive communication requests with the system environment. This result indicates that 22% of the requests leading to the activation of the load balancer are of the traditional type. In this element, the ExaLB manager unit is activated as soon as a request occurs that requires calling the load balancer. In this unit, the RSPRS/CPRPS Rewriter unit decides on the type of request (whether it is a standard or a dynamic and interactive request) based on the type of process request, request history, time and location limitations, and dependency. As seen in Fig. 5, the dynamic and interactive process request diagram has the highest adaptation to the conventional request diagram, leading to the load balancer's activation.

The frequency of change of events leading to the dynamic and interactive process creation is almost the same as the rate of regular events leading to the activation of the load balancer, except for the times 11, 29, and 47. The matching between dynamic and interactive process creation requests with conventional requests that lead to the activation of the load balancer is due to changes like the process requests in processing element no. 34.

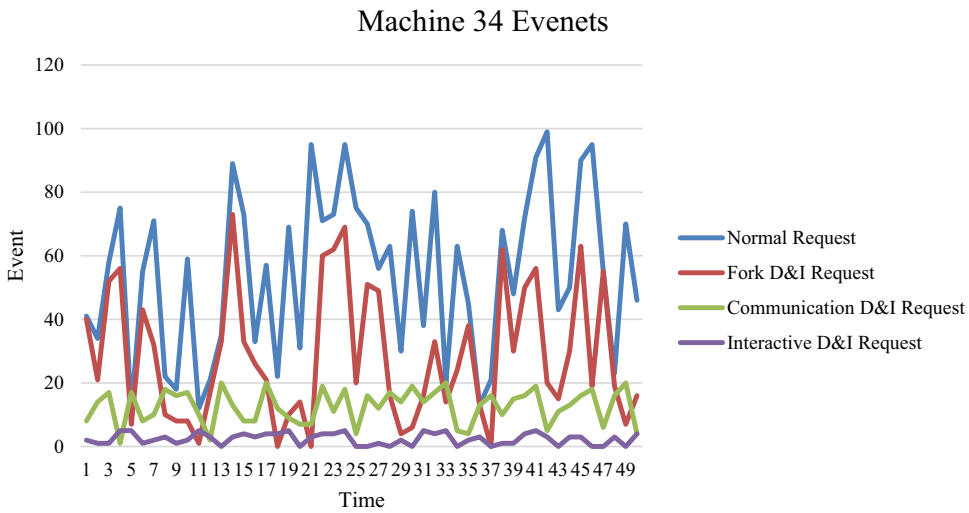

Fig. 5 Events leading to the activation of the ExaLB load balancer in processing element no. 34

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.622[a]	.387	.374	17.47062

Fig. 6 The correlation between events leading to the activation of the ExaLB and events leading to the dynamic and interactive process creations

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.019[a]	.000	-.020	6.44682

a. Predictors: (Constant), Request

Fig. 7 The correlation between the events leading to the activation of the ExaLB and the events leading to the dynamic and interactive inter-process communications

On average, in 50-time units, processing element no. Thirty-four were monitored, and 53.38% of standard process requests became dynamic and interactive process creation requests. The 53% occurrence of dynamic and interactive requests is the reason for the need for scalability of the computing system. The number 53% means that in 53 out of 100 requests leading to the activation of the load balancer, there is a need to create a new process that creates new global activity. This process expands the computing system to meet the requirements of the process running on element no. 34. It also means that 53% of the requested resources for the running process were not considered. This element creates a dynamic and interactive process that creates a new response structure through the Scalability + unit to meet the above requirements. Suppose the system is observed for a long time; then 53% of events lead to the activation of the load balancer into dynamic and interactive events of process creation. For a long time, the computing system has been scaled to the level where there would be little need to create new structures. The dynamic and interactive process creation events of ExaLB are used for scalability and the creation of new global activities to meet process requests. After a particular time, the size of the system by these processes reaches a stable level. In this situation, the system creates dynamic and interactive process events at a shallow rate.

In Fig. 5, the presence of zero points in the dynamic and interactive process diagram of processor creation means there is no need to create a new global activity to meet the processor's needs. Figure 6 shows the correlation status of dynamic and interactive process creation events and the requests leading to the activation of the ExaLB.

As seen in Fig. 6, the data of these two variables have a second-order correlation of 0.387 with each other, which indicates that if the system, especially element no. 34, if examined for a long time, these two variables would be less dependent on one another. This state indicates the independence of the computing system from scalability to meet the requirements of processes in processing element no. 34.

As seen in Fig. 5, about 19% of requests to activate the ExaLB result from dynamic and interactive communication requests between processes in the system. From the point of view of the ExaLB, this means that 19 out of 100 inter-processes communication requests are not considered in the basic structure of executing scientific applications. The number of inter-processes communication requests not considered in the initial response structure leans to zero faster than the process creation requests. If the computing system, and in particular processing element no. 34, is examined for a long time. According to the ExaLB, after the system reaches balance, the inter-processes communication resulting from the process requests reaches a stable state. This stability does not mean that inter-processes communication requests of dynamic and interactive nature do not occur. Instead, process requests not turn into dynamic and interactive inter-processes communication requests. This state is due to the consideration of two concepts of history and the behavior of each request by the ExaLB. Figure 7 shows the correlation coefficient between the number of requests leading to the activation of the ExaLB and the requests leading to the dynamic and interactive nature of inter-processes communication.

As seen in Fig. 7, the second-order correlation coefficient between these two variables tends to reach zero in the long run. This event indicates that if the system has reached stability over time, the number of requests leading to dynamic and interactive inter-process communications does not depend on the process requests. This event is due to the existence of request history, especially the request history for executing global activity, as well as request analysis, based on three-dimensional space of time, location, and dependency, which eventually leads to independence from the number of process requests. In such cases, the occurrence of dynamic and interactive inter-process events becomes dependent on Req_{Null}. From the point of view of the ExaLB, this means that the reasons for the occurrence of dynamic and interactive inter-process communication events in the state of balance are due to the formation of global activity. If the cause of global activity formation is to discover communications, the system expects dynamic and interactive inter-process communications to be formed in global activity.

Based on what has been said about the Scalability + unit, after the expansion of the computing system, the performance of ExaLB may decrease. As seen in Fig. 5, 6% of the requests that trigger the ExaLB are dynamic and interactive requests that require interaction with the system environment. However, the number of requests that lead to a dynamic and interactive request for interaction with the system environment is small compared to the other two types

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.002[a]	.000	-.021	25.68537

a. Predictors: (Constant), Request

Fig. 8 The correlation between events leading to the activation of the ExaLB and events leading to the dynamic and interactive request for interaction with the system environment

discussed. As a result, the rate of reduction of these requests and the independence of these requests to ones leading to activating the load balancer is higher than the other two types. From the point of view of ExaLB, when requests of dynamic and interactive nature for interaction with the environment are created, for responding to this type of request, ExaLB should call the resource discovery to expand the system and create a new response structure. On the other hand, according to what is mentioned in the Scalability + unit related to the ExaLB load balancer, the management system is not allowed to expand the computing system to any extent.

Consequently, the performance of the computing system may fall. This reduction in performance contrasts with the functional nature of the DECS. Figure 8 shows the correlation coefficient between the two variables of the number of requests leading to the activation of the ExaLB and dynamic and interactive requests for interaction with the environment.

As can be seen in Fig. 8, the second-order correlation coefficient between these two variables is zero. In DECS, after the formation of the region, it is expected that the load status pattern at the time of dynamic and interactive event occurrence for interaction with the system environment to be very similar to the load pattern when there are no dynamic and interactive requests. The difference in the R-value in Figs. 7 and 8 is also due to the concept of the variable α. The variable α causes the similarity of the distribution in the two states i and j to be the reason for the level of dependency of the number of D&I requests for interaction with the environment, to have a faster rate of becoming independent from events leading to activation of ExaLB**,** compared to the level of dependency of the number of D&I requests for inter-process communications. In this case, the D&I requests for interaction with the system environment depend only on the variable Req_{NULL}. Figure 9 shows the number of D&I requests responded to by processing element no. 34.

As can be seen in Fig. 9, the number of forks, inter-process communication, and interaction with environment D&I requests that have been answered by processing element no. Thirty-four are displayed. By comparing Figs. 5 and 9, it can be concluded that in terms of D&I requests, ExaLB has not been able to respond. Also, if the times in which there is no D&I request are not considered, the results show that ExaLB has responded to 53.70% of D&I fork requests and 44.99% of D&I inter-process communication requests, and 33.66% of D&I interaction with environment requests.

The lack of response to 46.30% of D&I fork requests is due to the inability to create the parameter $RI(t)$ in Eq. 17. As seen in Fig. 9, in all types of D&I requests, the response rate has increased as time has passed. This event is due to variables $Resource_{history}$ and $Req_{history}$. The more ExaLB has information about the status of resources and requests, the descriptors of request and resource (Eqs. 18 and 19) status would be more accurate.

The reason for the significant difference between the results of the D&I fork request with the other two types in Fig. 9 is due to $Req_{behavior}$ and the response mechanism of ExaLB to different types of D&I requests. ExaLB can use two different mechanisms for responding to D&I events, either by changing the functionality of the resource or by scalability. In these experiments, the mechanism used by the ExaLB is considered the scalability mechanism. When a fork-type D&I event occurred, the ExaLB, using the scalability mechanism, considering Eq. 22, created a new response structure for managing the event. The reason for the failure

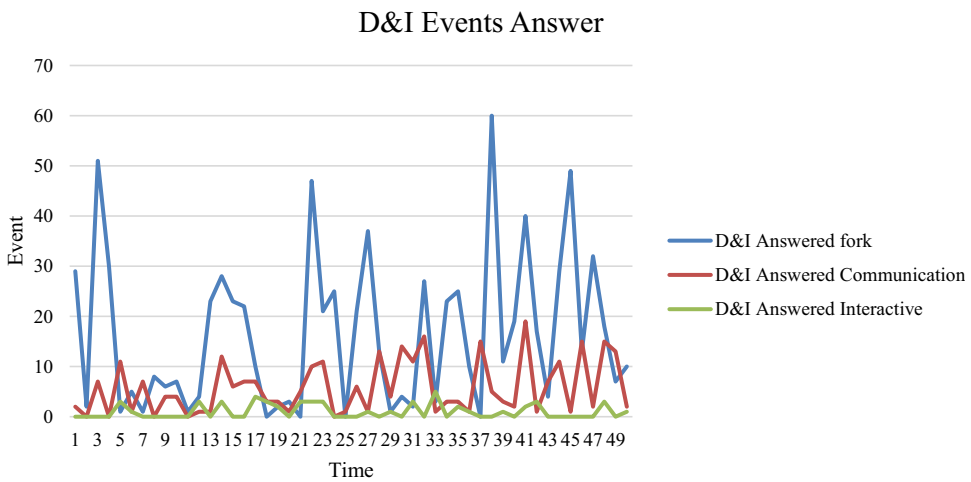

Fig. 9 Number of D&I requests responded by processing element no. 34

of the ExaLB, in the process of responding to the fork D&I requests in units such as 19, 22, 26, and 38, has been the need for the long time required to run the resource discovery.

In the 30^{th} time unit, the ExaLB unit successfully responded to 60 out of 62 fork D&I requests. The ExaLB load balancer has analyzed the behavior of requests in executing the application. This state, from the point of view of the ExaLB load balancer, means that there is enough time to extract the behavior of requests that occur in the 39th unit. In the 4th unit, 52 fork D&I requests occurred, of which the ExaLB answered 51 requests. By examining the status of the application in processing element No. 34, it is clear that the nature of the requests is such that they can be answered using scalability. The time required to execute the resource discovery for scalability and response to these requests, as well as the system's state after scalability, is acceptable to the ExaLB load balancer according to Eq. 22.

Events in the third time unit made it possible to state that the resource discovery, and its responsibility in responding to D&I requests, is effective over the functionality of the ExaLB. This event occurred when the request behavior had not been extracted or there had not been sufficient information about the functionality of the resource. Extracting request behavior and resource functionality and collecting more information about the status of the resources and requests reduces the dependency of the ExaLB on other system-managing units. Typically 50% of fork D&I requests cannot be answered by ExaLB's load balance. By examining the status of programs in number 34 and increasing the number of iterations of the experiment, it is observed that two factors are the most important reasons for the inability of the ExaLB load balancer to respond to these requests. The first factor is the inability of the resource discovery to respond to the scalability request in an acceptable time. The second factor is the lack of scalability due to reduced system performance (presented in Eq. 22). Examining the test results, especially the test results, in a situation where a fork-type D&I event occurs, but the ExaLB load balancer cannot respond. It made that in 38% of cases performing Eq. 22 is the reason for the inability to respond to the requests.

As can be seen in Fig. 9, the number of unanswered events by the ExaLB unit is higher than other types of D&I events when they were interactive D&I requests. The reason is that no element inside the computing system can respond to an interactive D&I request, and the resource managing unit must find it in the environment outside the system. There is no accurate information on which to form the matrices in Eqs. 15 and 16. Because such a processing element containing the resource has been outside the computing system, there is no information about its importance in global activities. When the resource is discovered, the ExaLB load balancer considers the importance of this resource to be 100 out of 100.

By examining Fig. 9, it can be seen that in time units such as 7, 28, and 43, the ExaLB load balancer has responded to all fork D&I requests. Except for a few time units, such as time unit 41, all events of interactive D&I nature are responded to, or it is entirely unable to respond. It was found that in the mentioned time units, the ExaLB load balancer encountered an event similar to similar events that had already happened. This repetition of events caused the ExaLB load balancer to obtain sufficient information about the request behavior (defined in Eq. 10) and the resource discovery pattern. The ExaLB load balancer, in time units, has extracted the description of the request and resource status using Eqs. 18 and 19, which leads to the inability to respond to events similar to the events of the mentioned time units. This event allowed the ExaLB load balancer to use a pattern of complete ability or inability for interactive D&I events.

By analyzing the ExaLB unit's functionality and the applications in processing element no. 34, it can be concluded that the main reason for such a phenomenon is not creating the descriptive matrix of the temporal changes for the desired resource as expressed in Eq. 18. The primary inability to create such a structure lies in resource discovery. Interactive D&I events strongly correlate with the load balancing function and the response history's data structures to similar events.

We are examining the scientific applications and the status of executing processes on processing unit no. 34, it can be concluded that the most influential variable on the ability or inability to respond to such events is the Null variables defined in the matrices of Eq. 5. As seen in Fig. 9, the ExaLB can respond to 45 out of 100 inter-process communication D&I events during system execution for a more extended period. The occurrence of these events from the point of view of the ExaLB load balancer means changing the reason for the request from the non-request status to the new request and changing the request behavior. From the resource analysis point of view, the inter-process communication resource, which had another role in the global activity, should be changed and proposed as a communication mechanism between the two processes. The response to inter-process D&I requests is such that scalable mechanisms are not used for the responding structure. Changing the functionality of resources and process is the method used to manage these events.

The need to use a separate model to respond to inter-process communication D&I events leads to time units such as 3, 5, 9, and 12, in which the ExaLB can not respond to the event due to the inability to change the role and functionality of the resource or processor request. The majority of times that the ExaLB does not respond to inter-process communication D&I requests are at system startup time. The ExaLB must be able to manage resources and processes by extracting the pattern of functionality and role by changing them when the inter-process communication type of D&I events

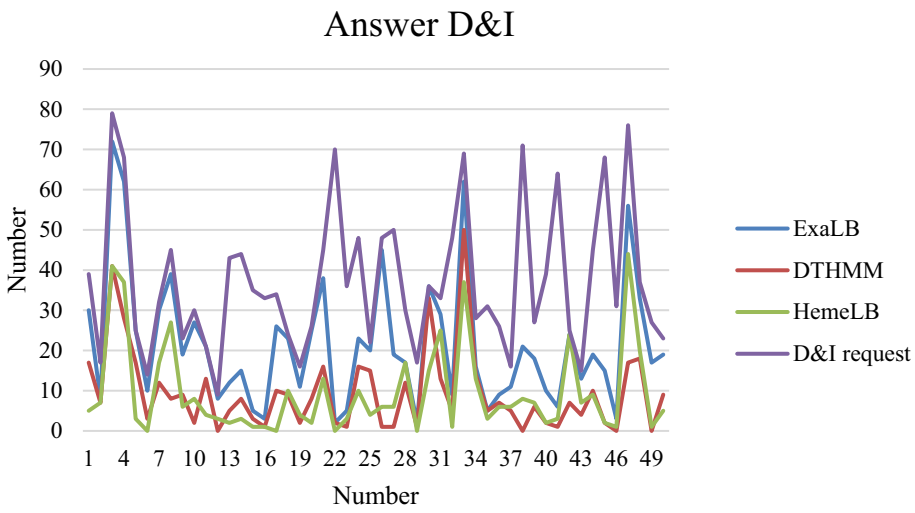

Fig. 10 Dynamic and interactive events affect the operation of load balancing in the ExaLB framework, ExaLB DTHMM, and HemeLB mechanisms

occur. Based on examining Fig. 9, it can be seen that most inter-process communication and D&I events responded to by the ExaLB management element is around the average line.

Figure 10 shows the number of dynamic and interactive events affecting the load balancing operation in the ExaLB framework, the DTHMM ExaLB mechanism, and the HemeLB mechanism.

As shown in Fig. 10, on average, every time the test is run, 37 requests are created, leading to load redistribution in computing element No. 37, which is somehow affected by dynamic and interactive events. This dynamic and interactive event can occur both in computing element number 37 and at any point of the computing system in a way that affects the implementation of the activities of the load balancer activated in computing element 37. On average, the HemeLB mechanism can manage the impacts of dynamic and interactive events on the load balancing operation in each test run five times. This issue is due to the primary condition of activation of the HemeLB mechanism in computing element No. 37 to manage the impacts of dynamic and interactive events on the operation of the resource load balancing. In a typical situation, in computing element number 37, the traditional load balancer is running. A dynamic and interactive event impacts the operation of the load balancer and causes the developed part of the load balancer based on the HemeLB to be activated. Based on the HemeLB mechanism, the load balancer makes a decision based on a set of indicators regarding the occurrence or lack of occurrence of the mentioned situation. By examining the experiments, it was found that the allocation model is not challenged in many cases due to the occurrence of dynamic and interactive events and their impact on the load balancing function. However, the collecting activities and the need to create expanded structures to manage the impacts of the dynamic event and interaction in the system and, consequently, the functionality of the distributed management unit have been affected. In the distributed load balancer based on the HemeLB mechanism, only one part of the functional tasks of the distributed load balancer is considered the focus of the effects of dynamic and interactive events on the operation of this element.

The distributed load balancer based on the DTHMM ExaLB mathematical model provides the ability for the distributed load balancer to manage the activity based on the changes that cause the function of the distributed load balancer to be violated, after analyzing each dynamic and interactive event. On average, in each test run, the mathematical mechanism of DTHMM ExaLB can detect nine events affecting the distributed load balancer's operation and management. Unlike the HemeLB, this mechanism considers all the situations. This information leads to the impact of the dynamic and interactive event on the operation of the load balancer. The distributed load balancer to dynamic and interactive events based on the DTHMM ExaLB mechanism can only handle nine out of 37 events. In this mechanism, the concept of the system state is used to describe the functionality of the distributed load balancer, and the return of the system state to a stable condition after the occurrence of a dynamic and interactive event is considered a method to manage the impacts of the dynamic and interactive event on the operation of the distributed load balancer—the mentioned mechanism to be unable to transfer the system to a stable state. If the distributed load balancer uses more variables than the traditional one to describe its state, creating the system state becomes complicated. Another challenge of this solution is the possibility of the inability to change the state of the system to a stable condition for the operation of the distributed load balancer.

As shown in Fig. 10, the ExaLB-based distributed load balancer manages 19 dynamic and interactive event occurrences affecting the operation of the distributed load balancer. The ExaLB management unit uses the two concepts of separation of dynamic and interactive events, as well as

the separation of the structures of the execution of the activities of the distributed load balancer after the occurrence of a dynamic and interactive event based on the framework introduced in this article. Using these two concepts makes the distributed load balancer based on ExaLB able to analyze the event and its impacts on the operation of the distributed load balancer based on the framework presented in Fig. 3 and the use of RSPRS/CPRPS. The lack of possibility in creating structures related to RSPRS/CPRPS due to the lack of detection of the incident causes the framework to be unable to utilize its management capabilities in 49% of the cases and experiments. This situation indicates that the framework introduced for the ExaLB-based distributed load balancer can manage the effects of dynamic and interactive events on the functionality of the distributed load balancer in 51% of cases.

8 Discussion

In DECS, the occurrence of a D&I by computing processes causes the response structure to change. Changing the response structure may cause the load balancer's pattern (or patterns) to be unable to respond and manage the new condition. The load balancer, as the central element of computing systems, assigns computing process requests to the appropriate resources during the execution of scientific applications. The purpose of the load balancer is to execute the allocation function, processes, and, ultimately, the scientific application in the shortest possible time. Two sets of CPRPS and RSPRS affect the functionality of the load balancer. A D&I event may cause the RI on which the load balancer redistributes the tasks to differ from the actual RI of the system. This paper introduces the ExaLB framework to address this challenge.

ExaLB framework changes the functionality of the load balancer, which used to be the mapping form of request space to the resource space, to the mapping form of the process status matrix to the source description space, and also decides on the type of request, whether it requires a standard distribution or a D&I request requiring redistribution. The ExaLB, unlike traditional load balancers, does not collect resources and process information while performing activities related to redistribution. The basic premise of ExaLB is that each process is part of global activity, so examining the process at a given moment cannot tell the characteristics of the process request. The development of the functionality of the ExaLB load balancer, based on Eq. 5, enables the ExaLB to retain information about the history, behavior, type, and reason for the request in global activity and related processes for each request. The development of Eq. 1 to Eq. 5 by the ExaLB causes the status of each resource to be analyzed at each moment. In traditional computing systems, the load balancer describes the resource status at the moment of redistribution based on the indicator(s). At the same time, the ExaLB assumes that the functionality of resources in DECS is determined in executing global activities. To analyze the resource, one must be able to analyze its history, functionality, and significance in global activities.

This event causes the ExaLB to use two matrices, $\left[Resource_{type} Resource_{fuctionality} Resource_{history} Resource_{NULL}\right]$ and $\left[Req_{type} Req_{Behavior} Req_{history} Req_{NULL}\right]$ to describe the functionality of the load balancer. Managing global computing processes based on the concept of global activity allows ExaLB to be able to gather the information needed to create the two matrices mentioned.

From the point of view of the ExaLB, each resource falls into one of four categories: computing and processing resources, memory resources, input and output resources, or file resources. Using Eq. 5 to define the functionality of the ExaLB provides the ability to make decisions about the nature of requests as well as the abilities of the resources. The matrix $\left[Req_{type} Req_{Behavior} Req_{history} Req_{NULL}\right]$ causes the ExaLB to use the Req_{NULL} the concept and Eqs. 9 and 10 determine whether the request is a D&I or a conventional request. If the request is D&I, it uses the introduced ExaLB framework. If the request made according to Eqs. 9 and 10 is conventional, it uses the standard distribution pattern of distributed computing systems (Khaneghah 2017).

The $\left[Resource_{type} Resource_{fuctionality} Resource_{history} Resource_{NULL}\right]$ allows the ExaLB to use $Resource_{NULL}$ Furthermore, Eqs. 15 and 16 decide whether or not this resource can respond to the D&I request. If the resource can respond to D&I requests, then ExaLB uses the resource mentioned in the global activity created to respond to D&I requests. This event is done by changing the concept of $Resource_{functionality}$ (Khaneghah and Sharifi 2014; Kayal 2009). This event allows the resource to modify its functionality according to the process requests so that it can respond based on the new function.

Equation 17 is a derivative description of Eq. 5. According to Eq. 5, the function of the load balancer is to map the matrices describing the status of the resource and the request to one another. Equation 17 states that resource and request status description matrices change during the execution of the scientific applications and, consequently, the activities related to the ExaLB. This attribute enables the ExaLB to be to perform redistribution-related activities when D&I requests occur based on Eq. 17 to extract $\frac{\partial \left[Req_{type} Req_{Behavior} Req_{history} Req_{NULL}\right]}{\partial_t}$ and $\frac{\partial \left[Resource_{type} Resource_{functionality} Resource_{history} Resource_{NULL}\right]}{\partial_t}$ matrices at any time during the execution of scientific applications. The mentioned matrices describing the status of the process and resource in terms of time are due to the inability of the

ExaLB, related to the need to retrieve information from resources and processes after a D&I event occurs. The difference in RI formed by data gathering of the load balancer and the actual RI of the system after D&I events occur is the main reason for the inability of the traditional load balancers to manage DECS. This issue has been resolved in the ExaLB framework by the Data Gathering unit based on Eqs. 18 and 19. Suppose the ExaLB can analyze the status of the resource based on time according to Eq. 18. In that case, it can decide whether it can form the resource description matrix of Eq. 17 based on the information and characteristics of the resource. Suppose the ExaLB can create a resource status description matrix in Eq. 17 based on Eq. 18. In that case, it can describe changes in resource type, history, functionality, and significance in global activities regarding time and resource status changes. If the ExaLB has the above information set, it knows the resource status and the variability of the four factors affecting the resource. This information makes the state of the ExaLB of the operating system equal to the state of the existing system during the occurrence of a D&I request. If the ExaLB element can analyze the request status based on time according to Eq. 19, the description of the system status from ExaLB's point of view should be the same as the actual system status.

The most critical function of Eqs. 18 and 19 is to enable the ExaLB to have accurate information about the status of processes and computing resources in a distributed system regarding the effects of the D&I events on them. The mentioned information set allows the load balancer to manage the redistribution activities in a way that includes the impacts of D&I requests on the entire system's status.

The ExaLB framework generates time-dependent RI using RSPRS / CPRPS Rewriter and Data Gathering units. The functionality of the ExaLB is not based on the RI of the activation time but on the RI, which is time-based and includes changes due to the occurrence of D&I events during the execution of redistribution activities. The use of Eqs. 18 and 19, as well as the description of the function of the system manager unit based on Eq. 5, causes the state of the system on which the load balancer is redistributing to match the state of the actual status of the computing system.

Funding The authors did not receive support from any organization for the submitted work.

Declarations

Conflict of interest All authors certify that they have no affiliations with or involvement in any organization or entity with any financial or non-financial interest in the subject matter or materials discussed in this manuscript.

References

Adibi, E., Khaneghah, E.M.: ExaRD: introducing a framework for empowerment of resource discovery to support distributed exascale computing systems with high consistency. Clust. Comput. **23**, 1–21 (2020)

Alowayyed, Saad et al. "Multiscale computing in the exascale era." Journal of Computational Science 22 (2017): 15–25.

Amelina, N., Fradkov, A., Jiang, Y., Vergados, D.J.: Approximate consensus in stochastic networks with application to load balancing. IEEE Trans. Inf. Theory **61**(4), 1739–1752 (2015)

Bakhishoff, U., et al.: DTHMM ExaLB: discrete-time hidden Markov model for load balancing in distributed exascale computing environment. Cogent Eng. **7**(1), 1743404 (2020)

Bok, K. et al.: Load Balancing with Load Threshold Adjustment in Structured P2P. In: 2018 IEEE International Conference on Big Data and Smart Computing (BigComp). IEEE (2018)

Chatterjee, Moumita, and S. K. Setua. "A new clustered load balancing approach for distributed systems." Computer, Communication, Control and Information Technology (C3IT), 2015 Third International Conference on. IEEE, 2015.

Domanal, S.G., Reddy, G.R.M.: Optimal load balancing in cloud computing by efficient utilization of virtual machines. In: Communication Systems and Networks (COMSNETS), 2014 Sixth International Conference on. IEEE, pp. 1–4 (2014)

Dongarra, J., et al.: The international exascale software project roadmap. Int. J. High Perform. Comput. Appl. **25**(1), 3–60 (2011)

Dongarra, J., Hittinger, J., Bell, J., Chacon, L., Falgout, R., Heroux, M. et al.: Applied Mathematics Research for Exascale Computing (No. LLNL-TR-651000). Lawrence Livermore National Lab (LLNL), Livermore, (2014)

Fiore, S., Bakhouya, M., Smari, W.W.: On the road to exascale: advances in high performance computing and simulations—an overview and editorial. Future Gen. Comput. Syst. **82**, 450–458 (2018)

Gharb, H., et al.: Challenges of execution trend in distributed exascale system. JDCS **1**(2), 140–151 (2019)

Ghomi, E.J., Rahmani, A.M., Qader, N.N.: Load-balancing algorithms in cloud computing: a survey. J. Netw. Comput. Appl. **88**, 50–71 (2017)

Heidsieck, G. et al.: Adaptive caching for data-intensive scientific workflows in the cloud. In: International Conference on Database and Expert Systems Applications. Springer, Cham (2019)

http://www.deep-project.EU. Accessed Oct 2022

Innocenti, Maria Elena et al. "Progress towards physics-based space weather forecasting with exascale computing." Advances in Engineering Software 111 (2017): 3–17.

Jain, S., Saxena, A.K.: A survey of load balancing challenges in cloud environment. In: 2016 International conference system modeling & advancement in research trends (SMART). IEEE (2016)

Jeannot, E., Mercier, G., Tessier, F.: Topology and affinity aware hierarchical and distributed load-balancing in Charm++. In: Communication Optimizations in HPC (COMHPC), International Workshop on, pp. 63–72 (2016)

Jiang, Y.: A survey of task allocation and load balancing in distributed systems. IEEE Trans. Parallel Distrib. Syst. **27**(2), 585–599 (2016)

Jyoti, A., Shrimali, M.: Dynamic provisioning of resources based on load balancing and service broker policy in cloud computing. Clust. Comput. **23**(1), 377–395 (2020)

Kayal, N.: The complexity of the annihilating polynomial. In: 2009 24th Annual IEEE conference on computational complexity. IEEE (2009)

Khan, S., et al.: Load balancing in grid computing: taxonomy, trends and opportunities. J. Netw. Comput. Appl. **88**, 99–111 (2017)
Khaneghah, E.M.: U.S. Patent No. 9,613,312. Washington, DC: U.S. Patent and Trademark Office (2017)
Khaneghah, E.M., Sharifi, M.: AMRC: an algebraic model for reconfiguration of high performance cluster computing systems at runtime. J. Supercomput. **67**(1), 1–30 (2014)
Khaneghah, E.M., ShowkatAbad, A.R., Ghahroodi, R.N.: Challenges of process migration to support distributed exascale computing environment. In: Proceedings of the 2018 7th International Conference on Software and Computer Applications (2018)
Khaneghah, E.M. et al.: Challenges of load balancing to support distributed exascale computing environment. In: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp) (2018)
Kołodziej, J., Khan, S.U., Wang, L., Kisiel-Dorohinicki, M., Madani, S.A., Niewiadomska-Szynkiewicz, E., et al.: Security, energy, and performance-aware resource allocation mechanisms for computational grids. Future Gen. Comput. Syst. **31**, 77–92 (2014)
Lehman, C., Nookala, P., Raicu I.: Scalable load-balancing concurrent queues in modern many-core architectures. In: SC19. ACM (2019)
Lieber, M., Gößner, K., & Nagel, W. E.: The potential of diffusive load balancing at large scale. In: Proceedings of the 23rd European MPI Users' Group Meeting. ACM, pp. 154–157 (2016)
Milani, A.S., Navimipour, N.J.: Load balancing mechanisms and techniques in the cloud environments: systematic literature review and future trends. J. Netw. Comput. Appl. **71**, 86–98 (2016)
Mirtaheri, S.L., Grandinetti, L.: Dynamic load balancing in distributed exascale computing systems. Clust. Comput. **20**(4), 3677–3689 (2017)
Mirtaheri, S.L., Khaneghah, E.M., Sharifi, M., Minaei-Bidgoli, B., Raahemi, B., Arab, M.N., Ardestani, A.S.: Four-dimensional model for describing the status of peers in peer-to-peer distributed systems. Turk. J. Electr. Eng. Comput. Sci. **21**(6), 1646–1664 (2013)
Mirtaheri, S.L., Khaneghah, E.M., Memaripour, A.S., Grandinetti, L., Sharifi, M., Bornaee, Z.: Multics and Plan 9: the big bangs in the distributed computing system universe. Comput. Sci. Eng. **16**(5), 76–85 (2014)
Mirtaheri, S.L. et al.: A mathematical model for empowerment of Beowulf clusters for exascale computing. In: 2013 International Conference on High Performance Computing & Simulation (HPCS) (2013)
Mondal, R.K., et al.: Load balancing on selected nodes with average tasks in cloud computing. J. Innov. Electron. Commun. Eng. **6**(2), 43–45 (2016)
Mondal, R.K., Ray, P., Sarddar, D.: Load balancing. Int. J. Res. Comput. Appl. Inf. Technol. **4**(1), 1–21 (2016)
Mondal, R.K. et al.: Load balancing with job switching in cloud computing network. In: Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications. Springer, Singapore (2017)
Mousavi Khaneghah, E., Noorabad Ghahroodi, R., Reyhani ShowkatAbad, A.: A mathematical multi-dimensional mechanism to improve process migration efficiency in peer-to-peer computing environments. Cogent Eng. **5**(1), 1458434 (2018)
Mukherjee, D., Borst, S. C., Van Leeuwaarden, J. S. H., & Whiting, P. A. (2016, March). Efficient load balancing in large-scale systems. In Information Science and Systems (CISS), 2016 Annual Conference on (pp. 384–389). IEEE.
Pate Ahmadian, A., et al.: Resource discovery in non-structured peer to peer grid systems using the shuffled frog leaping algorithm. JTEC **10**(4), 9–14 (2018)
Pourqasem, J.: Toward the optimization resource discovery service in grid systems: a survey. J. Appl. Res. Ind. Eng. **5**(4), 346–355 (2018)
Qureshi, M.B., Dehnavi, M.M., Min-Allah, N., Qureshi, M.S., Hussain, H., Rentifis, I., et al.: Survey on grid resource allocation mechanisms. J. Grid Comput. **12**(2), 399–441 (2014)
Ramezani, F., Lu, J., Hussain, F.K.: Task-based system load balancing in cloud computing using particle swarm optimization. Int. J. Parall. Progr. **42**(5), 739–754 (2014)
Rao, A. et al.: Load balancing in structured P2P systems. In: International Workshop on Peer-to-Peer Systems. Springer, Berlin (2003)
Rathore, N.K., Chana, I.: Job migration policies for grid environment. Wirel. Pers. Commun. **89**(1), 241–269 (2016)
Rathore, N.K., Rawat, U., Kulhari, S.C.: Efficient hybrid load balancing algorithm. Natl. Acad. Sci. Lett. **43**(2), 177–185 (2020)
Reylé, C., Richard, J., Cambrésy, L., Deleuil, M., Pécontal, E., Tresse, L.: Perspectives in numerical astrophysics: towards an exciting future in the exascale era. In: Proceedings of the Annual Meeting of the French Society of Astronomy & Astrophysics, pp. 133–137 (2016)
Shahrabi, Shirin et al. "Load Balancing in Distributed Exascale Computing Based on Process Requirements." Azerbaijan Journal of High Performance Computing, 2.1 (2018):158–167
Sharifi, M., Mirtaheri, S.L., Khaneghah, E.M.: A dynamic framework for integrated management of all types of resources in P2P systems. J. Supercomput. **52**(2), 149–170 (2010)
Straatsma, T.P., Antypas, K.B., Williams, T.J.: Exascale Scientific Applications: Scalability and Performance Portability. Chapman and Hall/CRC, Boca Raton (2017)
Teylo, L., et al.: A hybrid evolutionary algorithm for task scheduling and data assignment of data-intensive scientific workflows on clouds. Future Gen. Comput. Syst. **76**, 1–17 (2017)
Thakur, A., Goraya, M.S.: A taxonomic survey on load balancing in cloud. J. Netw. Comput. Appl. **98**, 43–57 (2017)
van Steen, M., Tanenbaum, A.S.: A brief introduction to distributed systems. Computing **98**(10), 967–1009 (2016)
Wang, K. et al.: Optimizing load balancing and data-locality with data-aware scheduling. In: 2014 IEEE International Conference on Big Data (Big Data). IEEE (2014)
Wang, Ke et al. "Load-balanced and locality-aware scheduling for data-intensive workloads at extreme scales." Concurrency and Computation: Practice and Experience 28.1 (2016a): 70–94.
Wang, K., et al.: Exploring the design tradeoffs for extreme-scale high-performance computing system software. IEEE Trans. Parall. Distrib. Syst. **27**(4), 1070–1084 (2016b)
Wylie, B.J.N.: Exascale potholes for HPC: Execution performance and variability analysis of the flagship application code HemeLB. In: 2020 IEEE/ACM International Workshop on HPC User Support Tools (HUST) and Workshop on Programming and Performance Visualization Tools (ProTools). IEEE (2020)

Faezeh Mollasalehi has a Masters of Computer Science from Shahed University and currently is a member of the operating system and network laboratory at this university. She is interested in the High Performance Computing Systems (HPC) and has been involved in several related research activities over the past few years, including in load balancing of the HPC such as grid, P2P, and exascale computing systems. Since 2015, she has been involved in a research project to improve the performance of load balancing in distributed peer-to-peer computing systems based on mathematical mechanisms. She is also interested to design and develop a load balancing for exascale computing systems.

Ehsan Mousavi Khaneghah is a faculty member of the Computer Engineering Department of Shahed University. His research interest is the design and development of distributed computing systems. He is researching the development of a distributed Exascale computing system. He had a patent called "PMamut: runtime flexible resource management framework in a scalable distributed system based on nature of the request, demand and supply, and federalism". U.S. Patent No. 9,613,312. 4 Apr. 2017". Which proposes a framework for managing the Distributed Exascale System. His favorite research fields are operating systems, Exascale systems, parallel and distributed systems, Cluster systems, Grid systems, P2P computing systems, applied mathematics, optimization, and e-commerce. He has successful experience in running industrial designs in high-performance computing systems. He is also a consultant of Master Plan designs in industrial areas like banks and industries, which need high-performance computing systems. Now, he is a member of the operating system and network laboratories of Shahed University.

Amirhosein Reyhani Showkatabad is an MSc Student with five years of experience working as a Researcher Assistant in High Performance Computing. Major interests are Process migration and load balancing in Distributed exascale computing environments. Furthermore, he has done plenty of projects based on Artificial Intelligence and Blockchain Technology. He has received B.Sc. in computer hardware and his MSc thesis is entitled: "Data Storage improvement for Scaling the Blockchain".

Seyed Alireza Seyednejad received a B.Sc. degree from Guilan University and an M.Sc. from the University of Tehran. He has worked on automated verification of HDL (Hardware Description Language) designs using artificial intelligence methods. Major interests include HDL design and verification, distributed systems and high performance computing techniques as well as artificial intelligence fields of study.

Faeze Gholamrezaie is a researcher in the field of Artificial Intelligence with a Master's degree in AI. She is dedicated to exploring the latest advances in machine learning and deep learning, and her innovative ideas have helped advance the field. Faezeh is constantly pushing the boundaries of what is possible and working on expanding our understanding of AI.