Cluster Computing (2020) 23:3349–3369
https://doi.org/10.1007/s10586-020-03091-5

ExaRD: introducing a framework for empowerment of resource discovery to support distributed exascale computing systems with high consistency

Elham Adibi[1] · Ehsan Mousavi Khaneghah[1]

Received: 7 July 2019 / Revised: 24 September 2019 / Accepted: 9 March 2020 / Published online: 23 March 2020

Abstract
In this paper, we introduced the framework to empowerment resource discovery units for supporting distributed exascale computing systems with high consistency. In addition to the execution of activities to find resources by ExaRD, this framework is able to manage and control the dynamic and interactive events in distributed exascale computing systems. For these reasons, the dynamic and interactive nature in distributed exascale computing systems is analyzed, based on which the impacts of the occurrence of the dynamic and interactive nature in computational processes on the functionality of ExaRD are examined. By analyzing the impacts of the dynamic and interactive concept on the functionality of ExaRD, decisions can be made for constituent elements of the ExaRD framework and its functionality. Using a two-dimensional framework of ExaRD to manage and control dynamic and interactive events in distributed exascale computing systems causes ExaRD to be able to be executed in traditional computing systems. This two-dimensional framework is also able to create responding structures outside of the computing system to respond to the necessities of the computational processes. The ExaRD framework redefines the functionalities function and generator space of RD. Our examination in terms of management framework indicated that this framework is able to manage and control dynamic and interactive events by 50 percent.

Keywords Distributed exascale computing · Resource discovery · Dynamic and interactive events · Framework · System state

1 Introduction

In high performance computing (HPC) systems such as grid computing, peer to peer computing and distributed exascale computing systems, the responding structure is completed during the execution of programs. Thus, computational processes of the global activity send new requests during the execution of a program [1]. If resources of the HPC systems can respond to the request, a load balancer unit allocates resources to the requester [2]; otherwise, the resource discovery (RD) is called. The resource discovery receives the request of the process under the existing limitations, and the analysis is based on the consideration of the time limitation, the type of the resource, and sometimes the location limitation [3]. The resource discovery should be able to find computing elements that contain the requested resource, and by resource sharing, the execution of the process activity can be continued. In such systems, the request of the process is not changing until the end of activities related to RD [4].

The main task of RD in traditional HPC systems like grid computing and peer to peer computing is searching the resource of the request and then making sure that features of the request of the process are compatible with those of the outside computing system [5]. The resource discovery should be able to find the best resource to continue the execution of the process in the shortest time [6].

As activities related to RD are performed outside the boundary and limitation of the resource management system, they contain more uncertainty compared to ordinary

✉ Ehsan Mousavi Khaneghah
EMousavi@Shahed.ac.ir

Elham Adibi
Elham.adibi@Shahed.ac.ir

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

activities that are performed by the resource management system [7]. When activities of RD are performed in an environment with a higher uncertainty (such as in a distributed exascale computing system), due to the impacts of the dynamic and interactive events, they have a different nature compared to those performed in a traditional HPC systems. In distributed exascale computing systems, in addition to the traditional constraints of the computing systems, boundaries and limitations related to the occurrence of the dynamic and interactive events by the computational processes are also defined [8]. Boundaries and limitations due to the dynamic and interactive nature of events create some new limitations for activities related to RD [9]. This necessitates having more accurate information about the request and its limitations, as well as having mechanisms to examine changes in the system.

To examine the performance of RD in distributed exascale computing systems, the framework of RD should be revised. The framework of RD in traditional HPC systems can only satisfy boundaries and limitations of the computing system [10], but it is unable to adequately satisfy boundaries and limitations related to the dynamic and interactive nature of the computational processes. Thus, in addition to the analysis of the impact of the dynamic and interactive nature on RD, a new framework should be introduced to manage challenges caused by the dynamic and interactive nature of the computational processes, thereby influencing on the functionality of RD. To create this framework, constituent elements of RD in traditional HPC systems should be revised, or new elements should be defined in a way that the revised or new elements have compatibility with the traditional HPC systems [11].

2 Related work

Grid computing acts as an architecture for executing HPC applications. This architecture presents sharing of resources in different geographical places [12]. In addition, the cooperative nature of the grid computing leads to the generation of the concept named virtual organization [13–15] which contains a dynamic set of data and resources which cooperate together to do a task [16, 17]. Due to the dynamic nature of the environment of the system, computational processes or resource features undergo changes during the execution of the program [18]. In [19], a framework was introduced (called Cactus) which is compatible with the structure of HPC systems whose resource characteristics change over time. In addition, a mechanism for selection of resources was introduced. If the performance of the system is beyond the boundaries, by this mechanism, permission of changing the allocation of resources through migration is given.

In [20], a peer to peer computing system was used as a platform for the dynamic and scalable RD. These systems efficiently manage dynamic of resources and directory services in the network. In peer to peer computing systems, computing elements are able to leave the system and join the system in any time. As such, the description map and features of the directory services are changing during the execution of the program. In this paper, the Twine architecture was introduced to perform and examine the scalable RD for which computing elements are able to send requests for resources without any limitation in location and content. To have accurate information of resources in this architecture, each of the computing elements updates periodically the information. If any of the resources does not update itself in the specific time period, it would be omitted from the network.

There are two methods for organizing computing elements in peer to peer computing systems which include the structured and unstructured methods [21, 22]. If the number of computing elements in the system is limited, a quick search and response to any query is guaranteed in the structured peer to peer systems. On the other hand, if information of resources in the computing system and outside of it changes dynamically, as the updated information should be distributed in the network and the system becomes reorganized, the performance of the network decreases, leading to the additional overhead. Thus, the structured peer to peer systems are suitable for maintenance and processing of resources that are static and does not change over time [23].

In the unstructured peer to peer systems for which a pre-existing structure is not defined for maintenance and identification of the place of information of the resources, the scalability is higher compared to the structured peer to peer systems [24, 25]. In such systems, there is no limitation in terms of location of computing elements and information of resources [26]. Due to the dynamic nature of the environment of such systems and the possibility of adding computing elements in any time, there are more variety of resources that computing elements are able to share with other elements. Distributing the updated information of resources in the whole network is a time-consuming task [27]. In [28], a full description of different mechanisms of RD based on the dynamic nature of the computing systems is provided.

A growing demand to computing resources and the need for changes in their features indicate the importance of RD. In [29, 30], limitations and challenges for sharing resources in the grid computing systems were discussed and a model was introduced which provides autonomy characteristic for computing elements in distributed computing systems. In the dynamic nature of the computing systems, reconfiguration of the system and creating a dynamic network are possible by the autonomy characteristic combined with self-organization.

In [31], behavior of each node in a large-scale system is examined. To do so, behavior of each node in different repetitions is observed in one loop. The work load of nodes participated in a small-scale system is compared with the workload of nodes participated in a large-scale system. Their results indicated that extrapolation of the workload is used effectively. In addition, the conducted simulation makes it possible to have information about the most number of nodes participating in a cluster system with a limited number of resources.

In [32], a method is presented to discover scalability bugs. This method is able to find scalability bugs and provides some methods to test the code. This method is implemented in several large-scale systems, which resulted in acceptable results.

2.1 Definition of generator space of resource discovery

One of the most important units of the resource management of the high performance computing systems is RD. By receiving a request from computational processes, RD finds the best resource in a logical time, making possible the continuation of the program [33]. After finding a machine containing the resource [34], permission of access to the requested resource is given to the requester [35].

necessities of the process requesting and characteristics of resources outside the computing system, while considering the responding time and the possibility of giving the permission of access to the resource [37].

Performance of RD depends on the Request Nature Set (RNS) and the Resource Attribute Set (RAS). During the execution of activities related to RD in traditional computing systems, the Request Nature Set does not change [38]. In such computing systems, RD tries to allocate a computing element to the process that its RAS is not changing during the response to the request of the process [41].

Any changes in RNS or RAS can be caused by the dynamic and interactive nature of the defined computational processes in the distributed exascale computing system, which may lead to failure of activities related to RD.

In distributed exascale computing systems, depending on the dynamic and interactive nature of computational processes, RNS of these computational processes may change during the response to the request of the process or during finding a resource. In addition, in such computing systems, depending on the nature of the distributed resource management system and definition of the local autonomy [39, 40], the frequency of changes related to RAS may increase.

Due to functionality of RD and two concepts of RAS and RNS, the generator space of RD can be defined as:

$$RD \overset{\text{Defined based on}}{\hat{∷}} \ll \overbrace{Process_{Request}, Resource_{out\ of\ system}}^{space} >, < \overbrace{finding, matching, permission - allocation, remove\ and\ add}^{operation} > , Prcess_{State}, Global_{activity} < Answer, True >> \quad (1)$$

In traditional HPC systems such as grid computing and peer to peer computing, the procedure of RD takes place in two ways: (1) based on conditions governed on the request and necessities of the process, RD makes queries from computing elements outside the computing system; (2) In specific time periods or after some events, information related to characteristics of resources outside the computing system are gathered, and when the process wants to have access to a resource that cannot be responded by the local computing system [36], by matching the request of the process with characteristics of the discovered resources, the right resource is allocated to the process. No matter whether finding the resource is taken place or the request is created, RD should be able to make consistency between

As can be seen in Eq. (1), definition of RD is based on the "$\text{Process}_{\text{Request}}$"and the "$\text{Resource}_{\text{out of system}}$". These two sets are the main definers of RD. In fact, RD is a function that maps the "$\text{Resource}_{\text{out of system}}$" to the "$\text{Resource}_{\text{out of system}}$". To do this mapping, RD should use four activities. First, RD should be conducted based on one of the aforementioned scenarios. Second, a resource that has 100 percent consistency with the request of the process, or a resource with the least similarities with the request of the process can be found by RD. This selection is determined based on the policy of RD [41]. In traditional computing systems, the 100 percent consistency is usually used because structure of the response is known. In distributed exascale computing systems, due to the unknown structure of the response, as well as the possibility of some changes in constrains of the request, the least similarity pattern can be used [42, 43]. Third, RD should have a permission to have access to the discovered resource. This permission can be total or partial. It can be also a function of time or be independent of time. In the total permission, RD allocates a discovered resource to the requester as part of local resources. In this case, the resource management system should provide the aforementioned transparency. In the partial permission, only some specific activities can be allowed for the requester, and the requester should create and manage a distinct activity (part of a global activity) in remote resources [44]. Forth, regarding permanent or temporary inclusion of computing elements containing the requested resource, decision should be made based on general policies of the resource management system. If the global activity is frequently running or the resource management is acting based on similarity of global activities, RD treats based on the permanent inclusion of computing elements. Otherwise, the treatment is based on the need and the request of the process.

Similarity of the global activity is used in computing systems in which the frequency of the execution of the global activity is low, but global activities that are executed by the resource management system have similarities [45]. Similarities are examined in terms such as procedure of the execution, beneficial computing elements and the request of the process.

Based on policies of the resource management system, RD should decide whether or not to reduce the unused computing elements of the system. This is one of the drawbacks of the distributed exascale computing system in comparison to the traditional HPC systems because it leads to increases in the execution time of the units of the resource management system [46].

Two fundamental concepts by which activities of RD are defined are the process state and the global activity. The first contains the cause of the creation and the nature of the request in the process, while the latter indicates whether the request is memory-oriented or not [47, 48].

Each activity of RD has two characteristics of "answer" and "true". The "answer" characteristic determines if RD is able to adequately respond to the request of the process. These two concepts are discussed in more details in the next subsection.

2.2 Failure of resource discovery procedure

In contrast to the load balancer, RD operates outside of the boundary and limitations of the system. Computing elements outside of the computing system do not contain constraints related to the response procedure. In addition, these elements are not obeying the governing rules of the computing system. Therefore, during the response procedure, RD may be confronted with a concept known as incapability of the response. Based on a criterion, RD should decide whether or not it is able to respond to the request of the process [3].

In computing systems, including traditional or distributed exascale computing systems, the creator of a request is a process. Each process contains a concept known as receiving of the response in the right time and location [49]. If the right time and location related to the necessities of the process is violated, there might be a problem in continuation of the life of the process. Thus, the "answer" characteristic of RD means a meaningful response to the request of the process. This concept is more complex in distributed exascale computing systems. In such systems, the occurrence of dynamic and interactive events during activities related to RD might change the concept of the meaningful response to the request. Thus, RD in such systems should use more accurate mechanisms to be able to provide an answer [50].

The "true" concept refers to the fact that the request should be correctly responded, which means RD does not fail during activities relate to the RD.

3 The dynamic and interactive nature of computational processes

The dynamic and interactive nature of computational processes is due to the fact that all variables governed on natural events and relationship of variables, as well as boundary and limitations of the system that are being examined are not known for scientists in a special field of science. The aim of the special field of science is using of the distributed exascale computing system based on knowledge of variables and their relationships, as well as boundary and limitations of the computing system [8, 51].

In distributed exascale computing systems, principle rules are scientific traditional programs that currently run on computing systems [52]. To recognize and discover the unknowns, a set of basic rules should be considered in the special field of science. These rules are implemented in the computing and processing systems over the past 50 years,

Resource management in traditional computing systems responses to requests based on processes considered in the initial design of the system [53]. In contrast, during the execution of a program and in response to requests in distributed exascale computing systems, executable elements with new processes are being created or a new relationship might be formed between the existing processes; thus unpredictable resources are requested from the existing processes. Such unpredictable requests that are created during the execution are due to the dynamic and interactive nature of the computational processes [54].

In contrast to the traditional computing systems, in distributed exascale computing systems requests with the dynamic and interactive nature are created which have not been considered at the time of designing the system [54]. This is due to discovery of the governed rules on natural events [48]. A new process might be created in the procedure of discovery of the governed rules on natural events, or a new relationship might be formed between processes that describe natural events. If boundaries of the system that considered for the discovery of natural events are not in accordance with the nature of events, a new relationship between processes describing events with computing elements out of the system is created [55].

In computing systems, in response to the request of the process, new requests are created or a new relationship is created between processes inside the system and the environment, leading to creation of a new process. In all of the aforementioned items, a request is formed in the computational process for which there is not defined any controlling and managing structure at the time of designing the system. The resource management should be able to create appropriate controlling and managing structure for the requests during the execution of the program.

By examining the nature of scientific programs that need distributed exascale computing systems [56], it can be concluded that in such applications, the frequency of the occurrence of requests that cannot be responded by the local system increases [54].

This is due to formation of requests with the dynamic and interactive nature and creation of a new global activity [48] for the response to them, as well as the fact that a controlling and managing structure to respond to such request is not defined [48, 57]. The dynamic and interactive event leads to creation of a new process with new relationship inside and outside of the system, the features that were not considered in the responding structure of the system at the time of its design.

3.1 The impact of the dynamic and interactive nature of computational processes on recourse discovery

The dynamic and interactive nature of computational processes influences RD in two ways [48]. First, the dynamic and interactive nature increases the frequency of calling RD, meaning that functionality of RD is changing. In traditional computing systems, RD is being called for finding a new resource which has the capability of the response to a request of the process. During the occurrence of the event with dynamic and interactive nature in computational processes, RD is used for the response outside of the boundary and limitations of the system [54]. The second influence of the dynamic and interactive nature on RD is related to RNS and the Request Imaging (RI). In traditional computing systems, if *Alpha* process in response to the request of *Beta* causes calling of RD, during finding appropriate resources, RNS related to *Alpha* process will not change.

3.2 The impact of the dynamic and interactive nature of computational processes on the Request Nature Set (RNS)

In this study, the Request Nature Set of the *Alpha* process is defined as Eq. (2):

$$RNS_{Alpha}(Beta) \overset{Defined\ based\ on}{::} < \overbrace{Request_{Nature}, Request_{type}, Request_{time}, Request_{location}, RAC_{Beta}}^{space} >, < \overbrace{Permission, Allocation}^{operation} >> \qquad (2)$$

As can be seen in the Eq. (2), RNS related to the *Beta* request is defined based on four spaces which include the nature of the request, the type of the request, and the time and location constraints of the request. In RNS, permission and allocation of a resource to the process can be defined.

In Eq. (2), the nature of a request is the cause of formation of *Beta* in the *Alpha* process. The *Alpha* process is part of a global activity. In traditional and distributed exascale computing systems, the global activity is a set of related activities which are responded by different elements of the computing system. The cause of formation of the global activity in distributed computing systems is due to the fact that an initial request existed in the computing

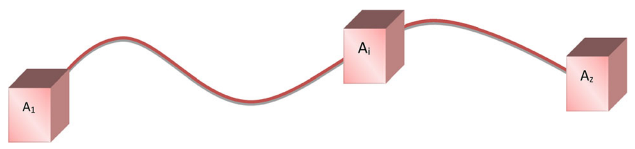


Fig. 1 A schematic of the global activity in distributed exascale computing systems [48]

element, part or all of which cannot be responded by the local computing element. To response to the request, the local computing system sends the request or part of the request to other computing elements of the system. Each computing element of the global activity is responding to part of the request. In Fig. 1, the global activity in distributed exascale computing systems is depicted [48].

As can be seen in Fig. 1, a request is formed in machine A_1 which has not the capability to respond to the request or part of the request. Based on the global activity mechanism, the request (or part of it) is transformed to another machine in the system. If each machine is considered equivalent to a point in the page, a line called global activity will be created, each point of which is a computing element that responds to the part of the request. The request is finished in machine A_z, meaning that all parts of the request are done by elements of the global activity. In the computing system shown in Fig. 1, the request is inside the system from A_1 to A_i. In machine A_i, there is no element in the computing system that can respond to the request (or part of the request). Therefore, RD is activated in machine A_i, and all machines from A_i to A_z are added to the system by RD [48].

The initial request that is formed in machine A_1 is called *Teta*. As *Teta* cannot be responded in machine A_1, it is transformed to machine A_2. If the concept of transformation in the system is being in such a way that during the transformation of *Teta*, the cause of the creation of the request and the reason that *Teta* cannot be responded is being transformed to A_2, *Teta* is memory oriented. Otherwise, if *Teta* initiates its activity as a local request in machine A_2 and has no information about its situation in machine A_1, it is called memory less.

If *Teta* is memory oriented, the cause of transformation and not responding to the previous computing element, as well as those parts that were responded in the previous element are kept. Thus, each computing element has accurate information about the nature of the request and the cause of its formation, as well as the reason that it is transformed. On the other, if *Teta* is memory less, during transformation from one computing element to another, it is considered as a new request that should be responded by the new computing element.

In Eq. (2), the type of the request is defined by the resource management system. For example, in [48], the type of the request lies in one (or a combination) of the four categories of I/O, file, process and memory. Any other categories of resources can be also used.

In Eq. (2), the time constraint of the request refers to the required time to respond to each request. This time either is determined by the computing element that creates the global activity or is determined during transformation of the request from one computing element to another.

In Eq. (2), location constraint of the request refers to the fact that whether response to the *Beta* request should be done in a specific location or not. In some computing elements, the *Beta* request should be finished in the element that the global activity is initiated. Thus, the resource management system should manage the global activity in such a way that leads to completion of the global activity in the element that it is initiated [58].

In Eq. (2), RAC_{Beta} denotes permission of access to a resource and allocation of the resource. For the *Beta* request, in machine number i, there should be a permission of access to the resource that has capability to respond to the *Beta* request (or part of it) in machine number $i + 1$. Permission of access and allocation of a resource to the request are granted by the load balancer if machine number $i + 1$ is in the local computing system [59], but are granted by RD if machine number $i + 1$ is outside the local computing system.

In traditional HPC systems, during the response to the *Beta* request, RNS is not changing by the load balancer and RD. This means that either RNS related to the *Beta* request is created in the computing element that creates the global activity with no changes during the global activity (if the request is memory oriented), or when the *Beta* request is transformed to the computing element, RNS is created by the resource management system which will be constant during the existence of the process containing the *Beta* request (if the request is memory less).

When a request is created in a system, each request is considered in the triple form as $<$ time, type, location $>$ ($<$ t, t, l $>$). When the *Beta* request is created in the *Alpha* process, either the load balancer or RD is called. At the time of calling, each of these two units considers the aforementioned triple form which is called the Request Image (RI). In traditional HPC systems, RI is constant during the response by any of these two units.

Based on system's theory, the *Beta* request is due to the interaction of the process with an element (a process or a resource). In traditional HPC systems, a new interaction does not occur and all the interactions can be defined in the structure of the initial response. On the other hand, in distributed exascale computing systems, creation of a process or relationship between processes inside or outside of the system creates a new interaction that has not been considered in the structure of the initial response.

3.3 The impact of the dynamic and interactive nature of computational processes on the *Beta* request

Based on the above discussion, it can be concluded that the dynamic and interactive nature of computational process leads to the formation of the *Beta* request, the RNS of which is different from that of the initial global activity. In other words, RI under the dynamic and interactive nature is changing during the execution of the load balancer or RD. In distributed exascale computing systems, the dynamic and interactive nature of computational processes leads to creation of a request that cannot be responded by the local computing system; thus it should be responded by RD.

In distributed exascale computing systems, RI is written as RI (t) which is due to the possibility of the occurrence of an event with the dynamic and interactive nature. In other words, when an event with the dynamic and interactive nature occurs, the time and location constraints, as well as the type of the requested resource might change.

As discussed above, in distributed exascale computing systems, creation of RNS or changing RI with time can be considered as the impact of the dynamic and interactive nature on the request of the processes that form the global activity. Formation of a new RNS or variability of RI with time means that RD or in some cases a load balancer should respond to them in such a way that mechanisms of RD can be used in distributed exascale computing systems.

4 Resource discovery in distributed exascale computing systems

The resource discovery in traditional HPC systems is searching for the requested resource based on constraints and limitations of the request. After discovery of the requested resource, the permission of access is given to the load balancer, while relationship between the requesting process and the process contains a resource is provided, and the requested resource is allocated to the requester. Such pattern for RD means that this unit needs to have four elements which include searching, adaptation, permission of access and allocation, while RD should also have connection with inter process communication management unit outside of the machine.

In traditional HPC systems, when the resource management system is activated, RD should be able to create RNS. In fact, the most important difference between traditional HPC systems and distributed exascale computing systems is related to the way of creation of RNS.

In traditional HPC systems, RNS is created based on the type of the request. In such computing systems, RD does not extract the nature of the request and does not gather information regarding the cause of the request in the computing element of the requester. Thus, activities related to RD are based on the type of the request and consideration of time and location constraints and constraints that are governed on RAC. This is due to the fact that the nature of the request is constant. In traditional HPC systems, the nature of the request is always determined at the time of designing the system, and structures of responses are based on the nature of the request. Defining different mechanisms for RD and selecting one mechanism to be used in computing systems indicate that the nature of the request is constant in traditional HPC systems.

In distributed exascale computing systems, the type of the request is only one of the constraints that affect RD, and the structure of the response related to RD is based on the nature of the request. As such, the resource management system or RD should obtain information about the nature of the request. This is due to possibility of definition of processes with the dynamic and interactive nature. In such processes, some events might be created that were not considered in structures of the initial response. In such systems, if selection of a mechanism for RD is only based on the type of the request, due to the dynamic and interactive nature of the computational processes, selected mechanisms might be invalid. Invalidity of the RD mechanism means either consecutive fails of RD or increasing the responding time of RD.

In distributed exascale computing systems, RD should be able to gather some information based on the nature of the request, and to create the structure of the response. Defining the structure of the response means selecting an appropriate mechanism based on the nature of the request for using in RD. In such computing systems, the nature of the request originates from a two dimensional space. Part of the nature of the request originates from the process requesting the resource and part of it originates from the global activity. In contrast to traditional HPC systems, actions and behavior of the process in distributed exascale computing systems are related to functionality of the process and activities that lead to creation of functionality of the process.

The functional space of the process in traditional HPC systems was also used. The functional space of the process refers to two subjects: (1) need; and (2) procedure of creating the need in the process.

The need concept implies examining the process at a specific time, meaning that at a specific time what kind of the resource is required for continuation of the activity. The procedure of creating a need in traditional HPC systems is based on structure of the data and functionality of the process as an abstract element.

In traditional HPC systems, all management activities of the resource management system are defined based on the process. Thus, the management unit in such systems is the process. On the other hand, due to the dynamic and interactive nature of computational processes, resource management in distributed exascale computing systems is related to the functionality and influence of a set of processes, and defines all of its management activities based on the global activity. Thus, the functional space of the process in distributed exascale computing systems only contains one dimensional space which contains need and for understanding features of the process, global activities are also considered.

Understanding the nature of a request in distributed exascale computing systems is only possible if the global activity concept and its status are considered. Thus, RD in distributed exascale computing systems considers each process as part of the global activity. Conditions governed on this global activity determine the basic part of the nature of the request. The nature of the request can be obtained based on vector algebra, as well as using the concept of global activity and its definition based on affine page and description of global activity based on Latin square [60].

In traditional HPC systems, it is assumed that if process O at $t = zeta$ requests to have access to a resource that cannot be responded by the resource management system, RD is activated. At $t = fi$ ($fi > zeta$), RD allocates the resource to the process O. During the time period *[zeta, fi]*, features of the process O and elements that can influence the process O do not change. This is due to the fact that $\text{Process}_{\text{state}}$ and global activity are both constant in Eq. 1, meaning that RI is constant during the period [*zeta, fi*].

Based on what already discussed, the dynamic and interactive nature of computational processes in distributed exascale computing systems cause changes in the $\text{Process}_{\text{state}}$, particularly changes in features of the request and global activity. As a result, RI changes during the period *[zeta, fi]*. Thus, the most important challenge for RD in distributed exascale computing systems is the fact that RI changes, which leads to failure of activities related to RD.

During the activities related to RD at the time period *[zeta, fi]*, RD should be informed about changes in the $\text{process}_{\text{State}}$ and global activity. Getting information about state of the process and global activity implies that a connection mechanism between RD, which was describing in Eq. 2, and $< \text{process}_{\text{State}}, \text{ global activity}_{\text{State}} >$, should be defined. Based on the above discussion and challenges associated with the dynamic and interactive nature in computational processes and functionality of the traditional RD, the framework shown in Fig. 2 can be applied for RD.

As can be seen in Fig. 2, RD in distributed exascale computing systems uses a two dimensional framework, which is in contrast to that in traditional HPC systems. In

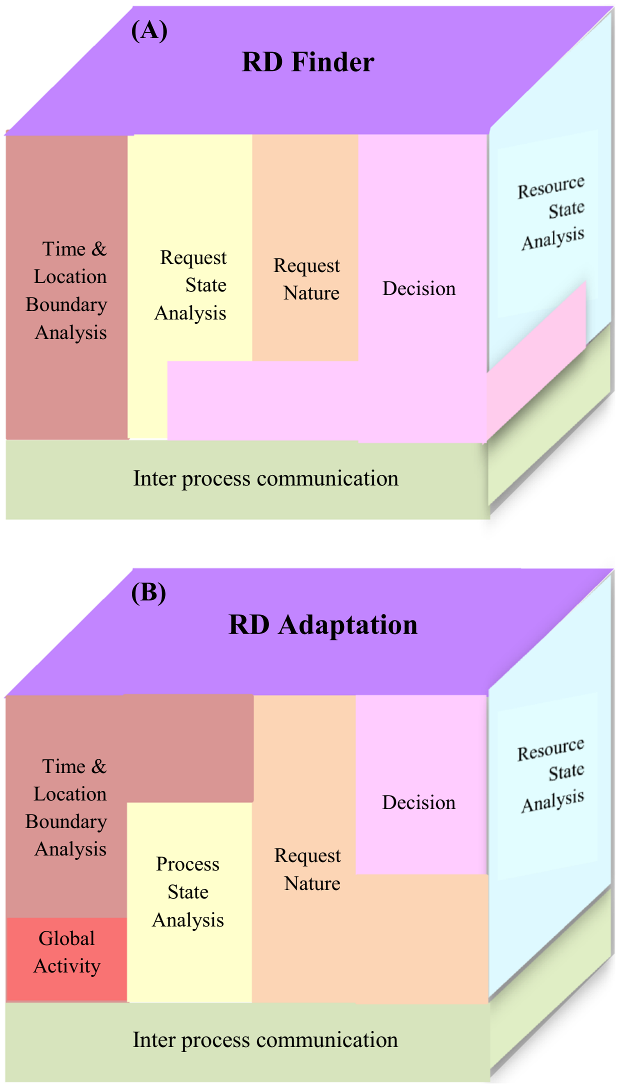


Fig. 2 The framework of (A) RD Finder and (B) RD Adaptation in distributed exascale computing systems

fact, in traditional HPC systems, RD assumes that there is no change in the RI elements during the period *[zeta, fi]*. A constant RI at the specified period means that the $\text{Request}_{\text{nature}}$ and RAC are constant. Thus, in such computing systems, after receiving the request of the process, RD leaves the computing element, meaning that RD becomes inactivated. The resource discovery acts based on any of the abovementioned mechanisms, and the decision is based on a set of information about computing elements. In addition, the decision is based on using indicators, and the accordance of RI with these indicators to find whether or not the computing element can respond to the process initiated RD. Thus, the framework presented in Fig. 2A can be used as a framework for traditional RD at which the $\text{Request}_{\text{nature}}$ and the $\text{Process}_{\text{state}}$ are constant.

Unpredicted changes in the $\text{Request}_{\text{state}}$ in such computing systems lead to failure of RD. In such computing systems, after making a decision, the load balancer allocates a resource to the process. Thus, part of the framework presented in Fig. 2B is also done by traditional RD.

A change in RI during the period *[zeta, fi]* causes changes in $\text{Request}_{\text{nature}}$ or RAC or both of them. On the view of RD, the period *[zeta, fi]* contains two periods *of [zeta, omega]* and *[omega, fi].* Omega is the time that RD discovers a resource. The occurrence of the dynamic and interactive nature in computational processes which leads to changing of RI, and consequently changes in the $\text{Request}_{\text{nature}}$ or RAC can take place in any of the two abovementioned periods.

determined by RD, activities related to the current RD stops and a new RD initiates.

Changes in the governed constraints on the request of the process mean changes in conditions of accessibility to the resource. In this situation, changing of the type of the resource and the constraints governed on it are not necessary. Thus, the occurrence of the dynamic and interactive nature causes changing of the pattern of the requesting process from the resource. As a result, the current RD stops. In addition, if the $\text{Request-}_{\text{nature}}$ changes, constraints and the type of the requested resource change. As RD uses a two dimensional framework, during the period from the start to the occurrence of the dynamic and interactive nature, RD might have examined computing elements for which the new conditions are valid. The reason of not using a temporal pattern for the state of changing the user request set (URS) is related to the fact that RD based on Eq. (4) does not search any information about the RAC.

$$\textit{ExaRD Function}: (\textit{Dynamic and Interactive}) \overset{\textit{Mapping}}{\leftrightarrows} \left(\textit{Change}_{RNS}(t), RI(t)\right) \tag{3}$$

As can be seen in Eq. (3), the impact of the dynamic and interactive nature on RD occurs in three ways: (1) temporal changes in RNS due to the occurrence of the dynamic and interactive nature on the requesting process or vice process; (2) a change in the state of RI which somehow influences on the requesting process; and (3) a change in constraints that are governed on RAC.

If the dynamic and interactive nature of the computational process occurs at the period *[zeta, omega],* as the dynamic and interactive nature is occurred during the activities related to RD, part of the RD unit which is in the machine initiating the RD activity (Fig. 2a) should somehow give these changes to $\text{RD}_{\text{Adaptation}}$ (Fig. 2b). The dynamic and interactive nature in the process of the requesting resource means changes in elements of Eq. (2). The resource discovery should analyze the dynamic and interactive nature of the occurred event, and then obtain the requesting parameters which should be given to $\text{RD}_{\text{Adaptation}}$ based on Eq. (4) by inter process communication mechanism outside the machine. As n units of time have passed from the start of RD, based on Eq. (4), it should decide whether to use the framework presented in Fig. 2 or to initiate a new RD activity.

$$DO = \sum_{Z=0}^{n}\left[(1+i)^{z} * X_Z\right] \tag{4}$$

In Eq. (4), DO is the rate of computing elements that are examined by RD, n is the time unit that RD is activated in the computing element, X_z is changes of the $\text{Request}_{\text{nature}}$ between the initial state and the new state after the occurrence of the dynamic and interactive event. If the value of DO exceeds from a specific limit that is

The occurrence of the dynamic and interactive nature of the computational processes in the two time periods leads to a change in the $\text{Request}_{\text{nature}}$, which itself necessities a connection mechanism between the two frameworks of RD presented in Fig. 2.

If RAC changes in response to changes in the $\text{Request}_{\text{nature}}$, during the second time period of the RD activities, RD certainly fails. If RD gathers information about each computing elements from the start as a form of < time, dependency, location > (< t, d, l >), then based on the framework presented in Fig. 2, failure of RD can be prevented. If as a result of changes in the $\text{Request}_{\text{nature}}$ during the first time period of the RD activities, then by changing elements of the $\text{Request}_{\text{state}}$ and the $\text{Process}_{\text{state}}$, compatibility between the two frameworks can be achieved. If as a result of a change in the $\text{Request}_{\text{nature}}$, only time and location constraints and the type of the resource change, no matter at which time RD is taking place, based on the framework shown in Fig. 2, RD should make compatibility between sections A and B [61, 62].

The consistency concept stated above means that machine A_0 that initiating RD finds machine P_0 which can respond to the request. Consistency between adaptation and finder is only meaningful if both of them are located over the same plane and mapping of each element of the finder and adoption element are matched. $A = f(p)$ in which $f(p)$ is an analytical function, can be written as Eq. (5):

$$\begin{aligned} A &= f(p) \\ &= \mathrm{RD}_{\mathrm{Finder}}(Request_{Nature}, RI, RAC) \\ &\quad + \mathrm{RD}_{\mathrm{Adaptation}}(Request_{Nature}, RI, RAC) \end{aligned} \tag{5}$$

In Eq. (5), *f (p)* is the functionality function of RD. This function is a complex-valued function in which its real part contains the finding part and its imaginary part contains the adapting part of RD. In Eq. (5), i indicates changes in the new request after the occurrence of the dynamic and interactive nature for the request based on which the RD activity is initiated. The imaginary variable (i) indicate that in distributed exascale computing systems if an event with the dynamic and interactive nature does not occur, or if it does, the nature of the created request is being the same as the previous request, the adoption part is not going to be activated.

Assume that RD in space K can be analyzed (space K is equal to the environment of the distributed exascale computing system). In this condition, Eq. (6) should be valid in order to have RD in space K with functional capability.

$$\left[\left(\frac{\partial \mathrm{RD}_{\mathrm{Finder}}}{\partial Request_{Nature}}\right) = \left(\frac{\partial \mathrm{RD}_{\mathrm{Adaptation}}}{\partial (RI, RAC)}\right)\right] \text{and} \left[\left(\frac{\partial \mathrm{RD}_{\mathrm{Finder}}}{\partial (RI, RAC)}\right) = \left(\frac{\partial \mathrm{RD}_{\mathrm{Adaptation}}}{\partial Request_{Nature}}\right)\right] \tag{6}$$

Equation 6 states that RD in distributed exascale computing systems is only analytical and functional if the two introduced parts in the framework of Fig. 2 are identical in terms of three concepts of RI, RAC and the $\mathrm{Request}_{\mathrm{nature}}$. In view of the resource management system, this means that the two parts of ExaRD framework are consistent. The resource management system only accepts ExaRD in the computing system and provides analytical possibility for that if based on three indices of RI, RAC and the Request-$_{\mathrm{nature}}$ creates a common page between the two parts of ExaRD. In this condition, constituent parameters that create the common page are identical in both pages. In fact, two parts of ExaRD are finding a resource for a $\mathrm{Request}_{\mathrm{nature}}$ under the RI condition and the response constraints of RAC.

In Eq. (5), which is functionality function of the RD unit, $\mathrm{RD}_{\mathrm{Finder}}(\mathrm{R}equest_{Nature}, RI, RAC)$ defines a three dimensional space of RAC, RI and the $\mathrm{Request}_{\mathrm{nature}}$. Then from the start of the activation to the end of the execution of ExaRD, the $\mathrm{RD}_{\mathrm{Finder}}$ can be described as a triple set $\langle \mathrm{R}equest_{Nature}, RI, RAC\rangle$, in which the $\mathrm{Request}_{\mathrm{nature}}$ and RI are vectors and RAC is a scalar. The $\mathrm{Request}_{\mathrm{nature}}$ is a four dimensional vector as < Process, Memory, File, I/O > , in which each vector at the start of the global activity is at the same direction of the unit vector and its value is equal to necessity of the global activity of the specific resource. If any of the computing elements can respond to parts of the request of the process that belongs to the global activity, direction of the $\mathrm{Request}_{\mathrm{nature}}$ vector changes. The allocated time to the process that belongs to the global activity to use a resource is equal to changes in the weight of the vector equivalent to the resource in the $\mathrm{Request}_{\mathrm{nature}}$ vector [63].

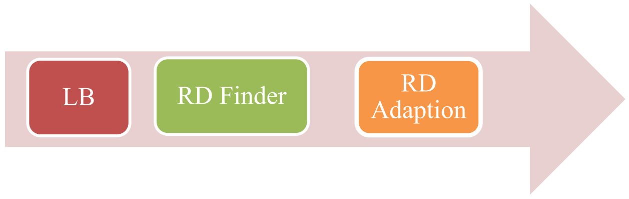


Fig. 3 Trend of the activation of the ExaRD unit

By connecting constituent points of the triple set of < $\mathrm{Request}_{\mathrm{nature}}$, RI, RAC > , the trend line of the execution of $\mathrm{RD}_{\mathrm{Finder}}$ in the specified three dimensional space for each $\mathrm{RD}_{\mathrm{Finder}}$ can be obtained. Similarity, a three dimensional space for the $\mathrm{RD}_{\mathrm{Adaption}}$ can be defined, based on which the trend line of the execution of $\mathrm{RD}_{\mathrm{Adaption}}$ can be obtained. The function in Eq. 5 or the functionality function of RD can define those terms specified in Eq. 6, through which a correspondence between points of the space of the $\mathrm{RD}_{\mathrm{finder}}$ and those of $\mathrm{RD}_{\mathrm{Adaption}}$ can be established, which implies that characteristics of the space of the requesting process is equivalent to characteristics of the space of the response. In view of RD, this means that activities related to RD has been successful.

$$\begin{aligned} &RD_{finder_{space}} = RD_{finder}(\mathrm{R}equest_{Nature}, RI, RAC)\ and \\ &RD_{Apation_{space}} = RD_{Apation}(\mathrm{R}equest_{Nature}, RI, RAC) \end{aligned} \tag{7}$$

If Eq. (5) can establish a conformal map between the two spaces of the $\mathrm{RD}_{\mathrm{finder}}$ and the $\mathrm{RD}_{\mathrm{Adaption}}$ based on Eq. (7), map of each page or each line from the space of the $\mathrm{RD}_{\mathrm{finder}}$ to the $\mathrm{RD}_{\mathrm{Adaption}}$ both in magnitude and direction is identical. The identical here means that the request has been successfully responded. In Fig. 3, events that lead to activation of ExaRD are shown.

As can be seen in Fig. 3, after creation of the *Beta* request, the load balancer in distributed exascale computing systems becomes activated. Based on the nature of the request, the load balancer makes decision if the *Beta* request is ordinary or it is a request with the dynamic and interactive nature. In addition, based on its mechanisms, the load balancer makes a decision about the capability of the distributed exascale computing system to respond to the *Beta* request. If the load balancer finds that the request is dynamic and interactive, and the distributed exascale computing system cannot respond to the request, ExaRD will be called. Activation of ExaRD implies that the $\mathrm{RD}_{\mathrm{finder}}$ is activated (Fig. 2a).

The $\mathrm{RD}_{\mathrm{finder}}$ is activated in a machine at which the request of ExaRD is located. As a result of the activation of the $\mathrm{RD}_{\mathrm{finder}}$ in the machine, the $\mathrm{Request}_{\mathrm{nature}}$ gets information from the load balancer to analyze the nature of the request. Analyses taken in the load balancer in terms of the nature of the request, including the type of the dynamic and interactive nature and analyses related to the data structure of the process that take place by the load balancer, are given to the Request$_{nature}$. This information is used as the basic information for creation of RNS. As a result of activation of the Request$_{nature}$, the Request$_{state}$ and time and location boundary analysis are activated in parallel. The Request$_{state}$ element analyzes the data structure of the process that has the *Beta* request.

With analysis of the data structure of the process with the *Beta* request, ExaRD can obtain the initial values of the request dependency and the type of the request. Although the type of the request can be also given from the Request$_{nature}$, if the process request is in the composite form and as a function of time, the obtained information about the type of the request from the Request$_{nature}$ do not contain information about the time dependent function, and only contains general information about the nature of the request of the process. ExaRD makes a decision based on the analyzed information by the Request$_{state}$ element about the type of the request and whether it changes overtime or not. This element also makes a decision based on the requests that are related to the *Beta* request and its required resource, as well as based on the requests that affect the *Beta* request and its required resource. This causes ExaRD to be able to make a decision in case of failure of the activities related to RD. Obtaining information about processes that affect (or affected by) the process that contain the *Beta* request causes RD to get information about dependencies of other processes or dependencies of the process containing the *Beta* request. Based on this information, the structure of the response can be defined, which include constraints, boundary and limitations and benefits of the dependencies.

The time and location boundary analysis element makes a decision about the time and location constraints of the process containing the *Beta* request. These constraints indicate that in which time and location, the response to the *Beta* request is acceptable for the process containing the *Beta* request, and in which conditions it is not acceptable. Information of the time and location boundary analysis element contains constraints that govern on RD. If ExaRD cannot satisfy the constraints, the procedure of RD fails. Information of this element causes that in the data structure of RNS, the initial values to be given to the request time and the request location. The Request$_{nature}$, time and location boundary analysis and the Request$_{state}$ causes initial values to be given to the data structure of RNS. The data structure of RNS in ExaRD is as a linked list, and for each computing element examined by the RD$_{Finder}$ unit, the first four variables of it and for each responding element, all six variables are being given values.

In distributed exascale computing systems, information about the Request$_{state}$, time and location boundary analysis and the Request$_{nature}$ elements are functions of time. When RD finds a resource for which constraints, limitations and the governed conditions are similar to those of the *Beta* request, RD creates a process called vice_RD in the computing element containing the requested resource. This process is the owner of the requested resource and the vice owner of the request. Thus, vice_RD is being informed about any changes in the request. Creation of the vice_RD process takes place when all variables of the RNS data structure have been given some values. Giving values to the data structure of RNS should contain the permission of access and allocation of a resource that the vice_RD is its owner.

Part of the data structure related to the constraints and limitations in Eq. 2 are initialized by the information of the time and location boundary analysis element. Information defined in this equation is compared against the data structure of the vice_RD process, and if they are the same, it can be concluded that the dynamic and interactive event is not occurred in the *Beta* request. If they are not the same, it can be concluded that the dynamic and interactive event occurred in the *Beta* request, causing activation of the RD$_{Adaptation}$.

The RD$_{Finder}$ and the RD$_{Adaptation}$ units interact with the Resouce$_{state}$ element. The Resouce$_{state}$ element which is in direct relationship with the Request$_{nature}$ knows about the state of the resource of the computing elements before and after the occurrence of the dynamic and interactive nature in the computational processes. Values are given to the Resouce$_{state}$ based on information of the operating system. This implies that RD in each computing elements obtains the state of resources that their type is the same as the one initially defined by the Request$_{nature}$. This information is given to the Resouce$_{state}$ element. This element contains information about existing resources in computing elements, no matter what constraints are governed on the *Beta* request. Based on information in the Request$_{state}$, time and location boundary analysis, and Resource$_{state}$, the RD$_{Finder}$ unit makes a decision regarding the fact that which resource in the computing element can respond to the process containing the *Beta* request.

The time period between the start of RD and activation of the vice_RD process is in the range of *[zeta, omega]*. In this time range, the occurrence of the dynamic and interactive nature in the process containing the *Beta* request can be discussed in two areas of RAC and the Request$_{nature}$. In view of ExaRD, the occurrence of the dynamic and interactive nature in RAC means that the permission of access to the resource is changing or the allocation pattern of the resource of the vice_RD is changing. Thus, the RD$_{Finder}$ sends a request to the vice_RD process to change the permission of access to a resource or allocation of a resource. If the vice_RD process can execute the request, by changing the structure of RNS and synchronizing it with the $\mathrm{RD_{Finder}}$, manages the dynamic and interactive event. If the nature of the request changes, the $\mathrm{RD_{Adaptation}}$ activates. In the $\mathrm{RD_{Adaptation}}$ unit, information about time and location boundary analysis and the $\mathrm{Request_{nature}}$ elements and part of information of the $\mathrm{Resouce_{state}}$ are received from the $\mathrm{RD_{Finder}}$ unit. Part of the information of the $\mathrm{Resouce_{state}}$ is received from the vice_RD process. The vice_RD process gets this information from the data structure of the resource of its own. In the $\mathrm{RD_{Adaptation}}$ unit, the $\mathrm{Process_{state}}$ element represents the vice_RD process. Any changes in the vice_RD and changes of *Beta* and its effects on the resource are analyzed by this element. Analysis by the vice_RD process contains analysis of the *Beta* request and the responding resource. As a result of this analysis, he $\mathrm{Process_{state}}$ element creates an allocation mapping. For allocation, through maintaining the allocation mapping mechanism or changing the allocated elements [8], the $\mathrm{Process_{state}}$ element manages the vice_RD process when the dynamic and interactive nature of the computational processes causes changes in the nature of the request.

$$\left\langle \left[Request_{State}(t,d,l), Process_{State}(t,d,l)\right] \overset{\text{Defined based on}}{\overset{\frown}{\because}} \langle RD, Finding\ condition\rangle \overset{\text{Defined based on}}{\overset{\frown}{\because}} IPC \right\rangle \tag{8}$$

After the occurrence of the dynamic and interactive event, the $\mathrm{RD_{Adaptation}}$ unit is activated and then a direct connection between the $\mathrm{RD_{Adaptation}}$ unit and vice_RD process is established. By the information of the vice_RD process, the $\mathrm{RD_{Adaptation}}$ unit provides the required consistency between the requesting process of the resource and the process containing the resource.

The request *REQ* with the time and location constraints causes an activation of RD. Based on the traditional definition of RD, the $\mathrm{RD_{Finder}}$ takes place based on two concepts on the $\mathrm{Request_{state}}$ and the $\mathrm{Process_{state}}$. The $\mathrm{Request_{state}}$ is the state of the request relative to RD and the $\mathrm{Process_{state}}$ is the state of the requesting process of the resource relative to RD.

In traditional HPC systems, the $\mathrm{Request_{state}}$ and the $\mathrm{Process_{state}}$ do not change during RD. On the other hand, in distributed exascale computing systems, these concepts depend on time, location and the process or the request relative to the global activity, as well as other global activities. Thus, Eq. (8) can be used as a controlling and managing mechanism for RD in distributed exascale computing systems.

As can be seen in Eq. (8), in contrast to the traditional HPC systems, two concepts of $\mathrm{Request_{state}}$ and $\mathrm{Process_{state}}$ are considered as functions of time, dependency and location. The dependency concept here is any kind of constraint other than time and location that govern on the space of the request and process. Due to the time concept that exists in such computing systems, the space of the request and the process might contain constraints to other concepts (for example, the other process or the other request).

If RD examines the following (a) and (b) conditions, it decides whether or not the computing element can respond to the request. Based on Eq. (8), (a) if the User Request Set (URS) is being equal to necessities of the process, for which responding to its request in the local computing system is not possible and URS is in the form of < RAC, limitation time, limitation location > , in each computing element that is being examined, its URS is re-examined with the requesting process or its vice. This is referred to as the *double verification of RNS*; (b) the RD mechanism can examine the condition stated in Eq. 9 during examination of URS related to a computing element.

$$if\ URS \equiv RNS \overset{\text{To somehow that}}{\overset{\frown}{\because}} RI(t)\ is\ constant \tag{9}$$

Equation 9 states that if URS of a computing element can satisfy the RNS related to the requesting process, the new computing element can only be added to the computing system by RD if it does not change the RI state of the system at the time of activities related to RD. If the RI state of the computing system changes in response to adding the new computing element, even if RNS can be satisfied by URS, due to the concept of the change chain, this computing element will be disregarded.

The start and end of the activity of RD are t = *zeta* and t = *zeta*$_2$, respectively. During *[zeta, zeta*$_2$*],* due to dynamic and interactive events, some changes occur in the computing system, making it as a dynamic system. A dynamical system is important in terms of changes related to elements that benefit from the requesting process or activities related to RD. As the system is in the dynamic and interactive state, RNS is changing. If at $t = zeta$, RNS becomes equal to $\mathrm{RNS}_{\mathrm{zata}}$, RD should find a resource in which while considering changes of RNS at *zeta* and $zeta_2$, RI would not change at *zeta* and $zeta_2$.

The concept of RI (t) concentrates on two concepts of the state of the system and creation of change chain. In a systematic view, RI is a function of an independent variable t, and its state can be changed due to the occurrence of a dynamic and interactive event either by the requesting process or other processes that affect the functionality of the requesting process. In view of change chain, RD that its URS is similar to RNS of the request, should not be in such a way that leads to chain of necessities or creation of a new necessity in the system.

5 Evaluation

To evaluate the suggested framework for RD in distributed exascale computing systems, a peer to peer distributed exascale computing system is used [48]. In this system, there are four types of resources that can be defined by the operating system. In such computing systems, the global activity can be considered as what shown in Fig. 1.

To evaluate the suggested framework, two global activities are executed by the system [48]. The Mesoscale Model Version 5 (MM5) [64] and the Weather Research and Forecasting (WRF) model [65] need high performance computing systems. Based on the global activity, each of these two models use computing resources that exist in the system. Thus, two global activities are being executed in the system. Each computing element of the system in each time can make a role in execution of one or two global activities.

The number of computing elements of the system is 120. With 120 computing elements, it would be possible to consider the computing system as a broad system for each of the models. The specified models are generally executed on fewer computing elements, such that their execution on 120 computing element helps us to analyze the condition of the models when they are over a broad system.

Due to the nature of the distributed exascale computing systems and the need to define the initial computing system, 50 computing elements are considered for the initial computing system. These 50 computing elements are consistent with the initial necessities of the computational processes related to scientific and application programs mentioned earlier. During execution of scientific and application programs, if computational processes need new resources to continue the execution, ExaRD extends the systems and adds new resources.

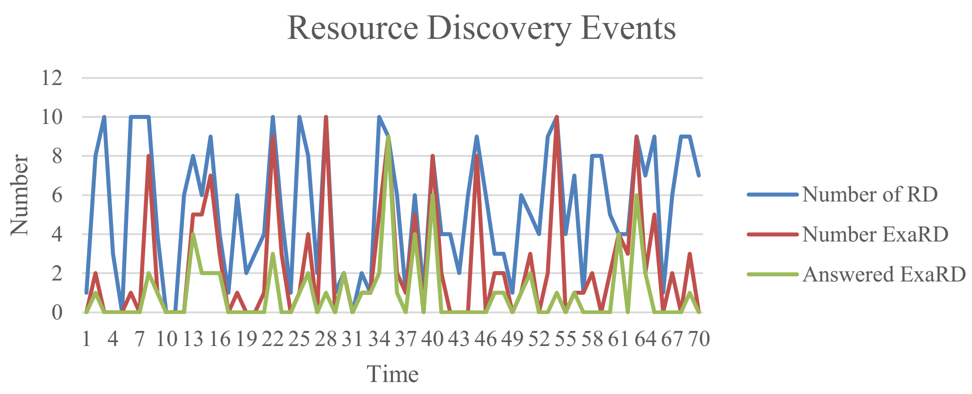
To create interactive and dynamic events on machine 12, a specific version of the software of the managing system [48] is used at which RD uses the ExaRD framework. Selection of machine 12 is due to the fact that in most of the times of the execution of scientific and application programs, this computing element participates in global activity of both software. Hardware configuration of this computing machine is equivalent to computing machines of the system. If each global activity stated in [60] are considered, in most of the times of the execution of the mentioned scientific and application programs, computing element 12 is the cross point of the two pages corresponding to global activities [66, 67]. Any other computing element can also be selected to be examined.

In computing element 12, the resource management system has been changed. As a result, the resource management system can manage three states of creation a process, relationship and interaction with the environment of the system that leads to the occurrence of a dynamic and interactive event [68, 69]. To this end, the resource management system in machine 12 (a) can manage processes that are not in the structure of the global activity; (b) manages the inter process communication between two processes that constitute the global activity that has not been considered in the initial structure of the global activity; (c) machine 12 is connected to another computing machine on which the two scientific applications exist. The resource management system of machine 12 can manage inter process communication related to the global activity and the corresponding process outside of the computing system which was not considered in the responding structure of the global activity. The system and the computing element 12 are examined in 70 time units. In Fig. 4, the number of RD, the number of calling ExaRD and the number of responses by ExaRD are shown.

As can be seen in Fig. 4, the load balancer in computing element 12 in each time encounters with 5 requests related to global activities that cannot respond to them and needs to call RD. the number of RDs in Fig. 4 indicates the number of events that lead to the call of ExaRD and RD in computing element 12. For example, at time 14, there are 8 requests for which there are no responses by the load balancer. Based on the nature of the requests and the fact that the basic element of the request is the $\mathrm{Request}_{\mathrm{state}}$ and the $\mathrm{Process}_{\mathrm{state}}$, from these 8 requests, five of them are responded by ExaRD unit and three of them by RD. All three requests related to the traditional RD are responded successfully, while out of the five requests related to ExaRD, four requests are responded successfully and one request has been failed.

As can be seen in Fig. 4, if computing element 12 is examined for a long time period, on average in each time, 5 requests which include traditional RD or ExaRD occur. Out

Fig. 4 The number of events that leads to calling of the RD unit

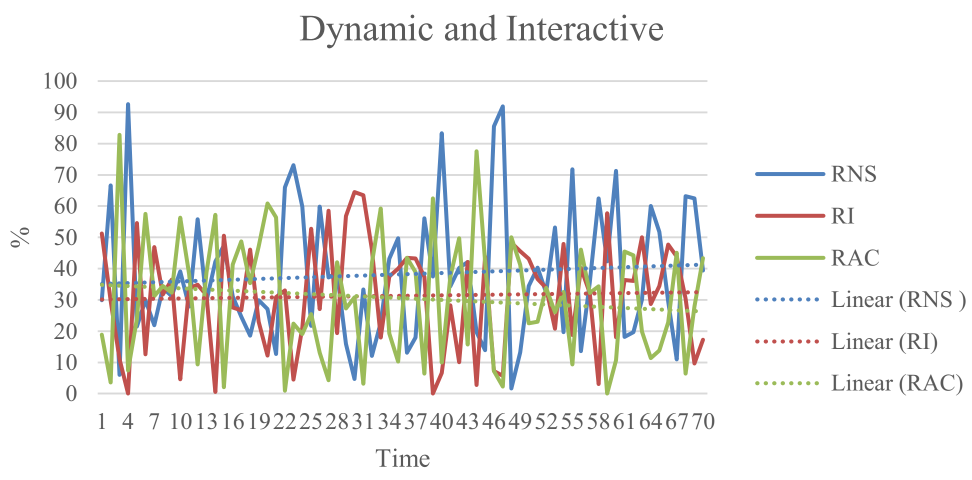


of these 5 requests, two of them are being sent to the ExaRD unit by the load balancer. Recognizing the dynamic and interactive requests by the load balancer is based on analysis of elements of the $\text{Request}_{\text{state}}$, the $\text{Process}_{\text{state}}$ and the global activity. If the $\text{Request}_{\text{state}}$ and the $\text{Process}_{\text{state}}$ are being in such a way that a request cannot be responded by the computing element 12, the load balancer sends the request to a traditional RD. If the analysis of the $\text{Request}_{\text{state}}$ and the $\text{Process}_{\text{state}}$ indicates that the request is dynamic and interactive which cannot be responded by the computing element 12, the request is being sent to ExaRD.

Analysis of the $\text{Request}_{\text{state}}$, the $\text{Process}_{\text{state}}$ and the global activity are being used in computing system [46]. In view of the load balancer, this implies that each request cannot be responded by the load balancer. From the analysis of the $\text{Request}_{\text{state}}$ and the $\text{Process}_{\text{state}}$, it can be determined that the request is dynamic and interactive or not. In experiments of this section, it is assumed that when the load balancer cannot respond to a request based on its own local resources, the request is being considered as a dynamic and interactive request and the load balancer tries to prove that the request is dynamic and interactive. If it is concluded that the request has a dynamic and interactive nature, the request is being sent to ExaRD. This leads to a reduction in the response time. In the second pattern that is not being used in experiments of this section, the load balancer can consider the request that cannot be responded by its local resources as an ordinary request and send it to a traditional RD. If it cannot be responded by the traditional RD, a request is being sent to ExaRD.

As can be seen in Fig. 4, when computing element 12 is being examined for a long time period, on average ExaRD in each time unit receives 2.27 dynamic and interactive requests and in each time unit responds to .9 of them. This implies that ExaRD can respond to 42.1 of the dynamic and interactive requests. As stated in Eq. 2, the existence of RAC and RNS for each request in a computing system for which RAC changes during the time period between finding the requested resource and resource allocation causes failure of RD. If the $\text{Request}_{\text{nature}}$ change due to a change in any of its constituents elements in Eq. 2, the ExaRD might change the $\text{Request}_{\text{state}}$ and its constituent elements. As stated earlier, if RAC changes between *omega* and *fi*, RD cannot respond to the request with a dynamic and interactive nature. In addition, definition of a threshold for variable DO in Eq. 4 is also another factor that contributes to failure of activities related to ExaRD. Thus, definition of the failure concept for functionality of ExaRD that stated in

Fig. 5 The impacts of RNS, RI and RAC variables on the dynamic and interactive nature



Eq. 3 causes the ExaRD cannot respond to all dynamic and interactive requests. In Fig. 5, the impact of each variables RNS, RI and RAC on dynamic and interactive events responded by ExaRD are shown.

It can be seen that changes of RI at a long time period causes the occurrence of 31 percent of events that during execution of activities related to ExaRD have a dynamic and interactive nature. The percentage impacts of RI and RAC on events that lead to the dynamic and interactive nature during the execution of activities related to ExaRD are very close. If computing element 12 is considered for a long time, the impact of RAC is about 30 percent. The difference between RI and RAC is in the curve slope during time. RAC uses a decreasing trend due to failure of activities related to ExaRD. If permission of access changes at the time between finding the requested resource and allocation of the, the ExaRD activity fails, and thus a change in RAC does not lead to a dynamic and interactive nature.

As stated in the ExaRD framework, in case of the occurrence of changes caused by RAC that lead to a dynamic and interactive nature, failure of activities related to RD at the specified time occurs. When peer to peer computing system reaches to equilibrium [48], the most changes of RAC also occur at this time period. As can be seen in Fig. 5, if the amplitude of the frequency of changes of RAC increases, the amplitude of the frequency of changes of RNS also increases. This is caused by a change in the permission of access required by the process and thus a change of the type of the request. This is occurred in experiments took place between 0 to 10 and also 40 to 50 time units. Change of RAC between *zeta* and *omega* time period impacts on dynamic and interactive events during execution of activities related to ExaRD. For example, at time units 4, 21 and 71, RAC causes 50 percent increase in the occurrence of dynamic and interactive events during execution of activities related to ExaRD.

As can be seen in Fig. 5 and also based on Eq. 3, the impact of the dynamic and interactive nature can be examined based on changes of RNS in time for the process requesting RD, a change of the RI state in such a way that causes a change of the $\text{Process}_{\text{state}}$ of the requesting RD or a change in constraints govern on RAC related to the requesting process.

As can be seen in Fig. 5, RNS has the most impact on the occurrence of dynamic and interactive events during the execution of activities related to ExaRD. On average, 38 percent of the causes of creation of the dynamic and interactive nature during ExaRD are caused due to changes in the nature or the type of the request. During 0 to 10 and 40 to 50 time units, changes related to RNS have an impact on the dynamic and interactive events during ExaRD. By examining scientific and application programs that are executing on the computing element 12, it is identified that at the abovementioned time units, processes that need ExaRD are changing the type of the requesting resource. As a result, with examining the trend of changes in RNS during a long time and their impact on the occurrence of the dynamic and interactive nature during an execution of RD activities, it can be identified that in 58 percent of the cases that experienced the most impact from RNS, the impact of RNS is due to a change in the type of the request.

In contrast to RI and RAC, the impact of changes in RNS during examination of the system follows an increasing pattern. By examining functionality of the scientific and application programs executing on the computing element 12, it is found that the increasing pattern is due to an increase of the number of events for which the nature of the request changes. A change of the nature of the request causes a need for creation of a new RD in the system which creates a dynamic and interactive nature during the execution of RD.

As can be seen in Fig. 5, RI uses the diagram with a decreasing slope which is caused by activation of RI. In initial experiments, activation of RI is due to variation of the time constraints, while after the time unit 28, a change of RI is due to changing of the dependency constraints. The frequency of the occurrence of changes of RI, and consequently its impact on events with the dynamic and interactive nature during ExaRD are reduced due to the impact of the location and dependency constraints. At time units 30 and 31, the impact of RI on events with the dynamic and interactive nature during execution of the ExaRD activities is about 60 percent. This is due to the fact that during the abovementioned time units, the time constraints that governed on a request by which ExaRD is activated are changing by processes.

Figure 6 shows the number of events that leads to the dynamic and interactive nature due to changes of RAC, the $\text{Request}_{\text{nature}}$ and the DO variable in the computing element 12 for the time unit 70. As can be seen, changes of RAC, the $\text{Request}_{\text{nature}}$ and the DO variable follow a unique pattern. As stated in the framework of ExaRD, changes of either RAC or the $\text{Request}_{\text{nature}}$ are the cause of the dynamic and interactive nature during RD that requires the use of a framework such as ExaRD or RD. Generally, the request change leads to the RAC change, although in the experiments, there are time units for which the request change is not accompanied with the RAC change. In units such as 58 to 61, although the request change is occurred, there is no change in RAC. By examining processes that need ExaRD during this time period, it can be found that the cause of the request change is changing of the nature of the request that leads to a request with a different nature, but the amount of the DO variable is not in such a way that to stop activities related to RD and create a new RD. In the

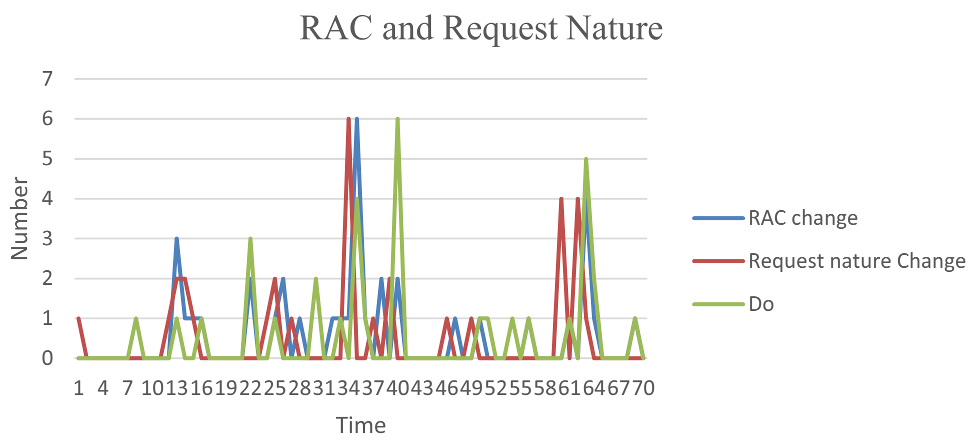


Fig. 6 Values of the RAC change, the Request nature change and the DO variables in the computing element 12

Table 1 Correlation between the responded requests by ExaRD and the RAC change

Model summary				
Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.907[a]	.823	.820	.71277

[a]Predictors: (Constant), RAC Change

Table 2 Correlation between the responded requests by ExaRD and the Request nature change

Model summary				
Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.035[a]	.001	− .013	1.69183

[a]Predictors: (Constant), Request Nature Change

Table 3 Correlation between the responded requests by ExaRD and the DO variable

Model summary				
Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.814[a]	.662	.657	.98394

[a]Predictors: (Constant), Do

conducted experiments, in 54 percent of the changes undertook by the RAC change, ExaRD can successfully respond to the requesting process.

In Table 1, correlation coefficient between the RAC change and the number of requests that are responded successfully by ExaRD are shown. The correlation coefficient of these two variables is .8, which indicates a high correlation. In the conducted experiments, in 54 percent of the cases, RD can respond to the request after the occurrence of the RAC change, implying that 54 percent of the RAC change occurs at the time period before finding the requested resource that can be managed by ExaRD.

Table 2 shows the correlation coefficient between the responded requests by ExaRD and the request nature change variable. As can be seen, the correlation coefficient between the responded requests by ExaRD and the request nature change variable is weak, implying that the $\text{Request}_{\text{nature}}$ change by ExaRD is responded to some extent. In contrast to the RAC change, RD in both time periods tries to respond the $\text{Request}_{\text{nature}}$ change, causing the $\text{Request}_{\text{nature}}$ change as a weak correlation with the responded requests by ExaRD.

Table 3 indicates that there is a moderate correlation between the responded requests by ExaRD and change of the DO variable. As can be seen in Table 3, there is a moderate correlation between the responded requests by ExaRD and changes in the DO variable. This is due to the fact that the DO variable examines whether as a result of changing the nature of the request, RD can be continued. By considering the number of examine elements, this variable examines whether changing the nature of the request can lead to continuation of RD. In implementation

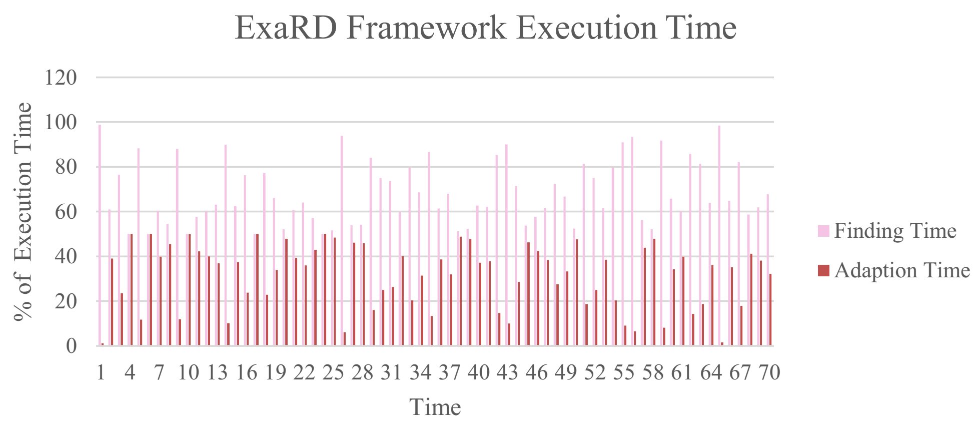


Fig. 7 Execution time for activities of the RD_{Finder} and $RD_{Adaptation}$ sections in ExaRD

of the distributed exascale computing systems [48], it is assumed that RD continues. Thus, calculation of the DO variable is used for cancellation of RD activities.

As can be seen in Fig. 6, the DO variable is equal to around 1, except in time units like 23, 36 and 64. This implies that based on the pattern in computing element 12, if the $Request_{nature}$ changes one unit, ExaRD stops activities related to RD. In order to decide about the value of the DO variable, ExaRD examines computing element 12 for specific time period and usually does several RD activities that do not fail. In Fig. 6, the computing system reaches to equilibrium from 0 to 40 time period. Thus, ExaRD considers the mean value 1 to calculate the amount of the DO variable.

As can be seen in Fig. 7, the required time for doing activities of the finding section of the suggested framework of ExaRD is higher than the required time for activities of the adoption section. In time units 5, 11, 18, 25, 39, 51 and 58, the time that is required for taking place the two units are nearly equal, which is caused by the fact that both RAC and the $Request_{nature}$ are changing simultaneously. The change of RAC takes place during the execution of RD_{Finder} to find a resource. In time units like 60, 57, 56, 36, 27, 15, 6, 1 and 68, the required time for adaptation is negligible compared to the required time for the RD_{Finder}. This can be due to the fact that in the above time units, specifically at time units 56 and 57, only the type of the nature of the request changes. On the other hand, at the above time units, the initial resource is found by ExaRD, but changes of the nature of the request causes execution of the adoption section.

6 Discussion

By redefining of the RD concept, the framework for ExaRD is introduced. This definition considers the fact that in spite of changes in RNS, RI and RAC over time due to the dynamic and interactive nature, the RD activities can still be done.

Based on Eq. (1), the occurrence of the dynamic and interactive nature leads to re-definition of effective elements and spaces on the RD. Equation 1 states that RD in which spaces and based on what conditions and execution of which activities can be re-defined. As ExaRD manages and executes activities related to RD at the time of the occurrence of the dynamic and interactive nature in the requesting process, this re-definition is based on the $Process_{state}$ and the $Request_{state}$.

Concepts such as the cause of the request, the type of the request, boundary and limitations of the request and responding to the request are based on the $Process_{state}$. In addition, concepts such as those activities that should be done by RD in order to respond to the request are based on the $Request_{state}$. Based on this definition, ExaRD can analyze two basic states of the requesting process relative to RD and the request relative to RD. On the other hand, in traditional HPC systems, RD only analyzes the $Resource_{state}$. Thus, definition of the functionality of ExaRD based on Eq. (1) causes the basic concept of the defining RD from the requesting resources changes to two concepts of the $Request_{state}$ and the $Process_{state}$.

The reason of re-defining of the basic element that defines RD and its consistency with two concepts of the $Request_{state}$ and $Process_{state}$ is due to consideration of the dynamic and interactive nature because it causes a change in the state of the requesting process relative to RD, and as this may happen during activities related to RD, the $Request_{state}$ related to RD should be analyzed.

During execution of activities related to RD, ExaRD evaluates the $Process_{state}$ requesting the resource and the created request. During the execution of the program, an event with the dynamic and interactive nature can make changes in the process containing the resource or ExaRD. For example, due to inter process communication, data structure of ExaRD is changing, such that it is looking for a resource that was not requested by the process. In this paper, the dynamic and interactive nature is only considered at a process requesting the resource. Thus, ExaRD can only respond to the request if changes in the request were established by a requesting process.

In ExaRD, definition of the RI space and its conversion from a constant value to a varying value with time cause consistency of the state of the real system with the state of the system that ExaRD has information about it and does activities related to ExaRD. To this end, ExaRD uses redefinition of the traditional RD activities based on the $\text{Request}_{\text{state}}$ and the $\text{Process}_{\text{state}}$.

Activities of ExaRD are related to the nature of the request and its response to the occurrence of the dynamic and interactive events and also the requesting $\text{Process}_{\text{state}}$ relative to RD. This re-definition of the basic element of the creation of RD causes ExaRD to use separate two dimensional frameworks. In this framework, two tasks of finding and adoption are separated. Activities related to traditional RD are considered in both sections of the framework. The reason that ExaRD is able to manage changes of RNS and RI with time is re-definition of activities of finding and adoption based on either the $\text{Request}_{\text{state}}$ or the $\text{Process}_{\text{state}}$. The finding section is redefined based on the request, which causes in the framework of ExaRD, the finding section can be able to consider constituent elements of Eq. (2) that are related to the $\text{Request}_{\text{state}}$.

By considering the time and location boundary and limitations of the request, decision is based on changes took place on the request due to the occurrence of the dynamic and interactive event. Decision is also based on the state of the discovered resource that is able to respond to requests based on RI (t). The decision is taken place by the $\text{RD}_{\text{Finder}}$ based on the $\text{Request}_{\text{state}}$. The structure of the $\text{RD}_{\text{Finder}}$ unit is such that it is in full interaction with the RNS structure mentioned in Eq. (2) and also the data structures of the $\text{Request}_{\text{state}}$ mentioned in Eq. (4). Using a linked list, structures of RNS and the $\text{Request}_{\text{state}}$ can provide analysis of initial conditions governed on the request, while maintaining information related to computing elements are examined by ExaRD and also information related to time and location boundary unit mentioned in Eq. (2).

The $\text{Request}_{\text{state}}$ element that is in full interaction with the $\text{Request}_{\text{nture}}$ element in distributed exascale computing system depends on time and location constraints and any other dependency. Information of this unit is given to the vice_RD process to complete data structure of the requesting process. As stated in Eq. (8), conditions governed on searching a resource are given to $\text{RD}_{\text{Finder}}$ by vice_RD.

The dynamic and interactive nature is such that consistency between the finding and adoption units should exist. In this condition, RD should be informed about changes of the requesting $\text{Process}_{\text{state}}$ and changes of RI and RNS by this process, such that this unit tries to find the requesting resource in the new state of the system. This happens when activities that have been taken place in finding and adoption are consistent. To this end, based on Eq. (4), ExaRD decides whether changes that took place are in some way that it is logical to continue the activity based on the current RD. This is a parametric decision and is different from a computing system to another. If it is valid, based on Eqs. (5–8), it can be decided if through the $\text{Request}_{\text{state}}$ and the $\text{Process}_{\text{state}}$ elements, consistency of the finding and adoption units can be established. The consistency is satisfied based on Eqs. (7 and 8). Based on the pattern used by ExaRD, challenges due to the dynamic and interactive nature during activities related to RD can be solved. This can be done by increasing empowerment of RD, while considering changes of RNS and RI with time which was examined in Eq. (3).

7 Conclusion

In this paper, while functionality of RD in distributed exascale computing systems is examined, a two dimensional framework is introduced to increase empowerment of RD to manage and control dynamic and interactive events that are created by the requesting process. The framework presented in the present study changes the basic activities of RD from the type of the request to the $\text{Request}_{\text{state}}$ and the $\text{Process}_{\text{state}}$. In this way, if the dynamic and interactive event occurs in the requesting process, through the gathered information, RD can control these conditions. By considering failure of the RD activities, it is discussed that in which conditions ExaRD can respond to the dynamic and interactive events.

Experiments conducted in distributed exascale computing systems indicated in which conditions the introduced framework can control and manage the dynamic and interactive events during the RD activity. If all constraints of the request are changing, the second part of the framework of ExaRD needs to spend an equal or even more time than that of the first part. By changing the data structure of the requesting process and the process containing the resource, ExaRD can manage the created conditions and can finish the RD operation.

References

1. Bogdanova, V.G., Bychkov, I.V., Korsukov, A.S., Oparin, G.A., Feoktistov, A.G.: Multiagent approach to controlling distributed computing in a cluster Grid system. J. Comput. Syst. Sci. Int. **53**(5), 713–722 (2014)
2. Banerjee, S., Hecker, J.P.: A multi-agent system approach to load-balancing and resource allocation for distributed computing. In: First Complex Systems Digital Campus World E-Conference 2015. Springer, Cham, pp. 41–54 (2017)
3. Navimipour, N.J., Rahmani, A.M., Navin, A.H., Hosseinzadeh, M.: Resource discovery mechanisms in grid systems: a survey. J. Netw. Comput. Appl. **41**, 389–410 (2014)
4. Xu, J., Lam, A.Y., Li, V.O.: Chemical reaction optimization for task scheduling in grid computing. IEEE Trans. Parallel Distrib. Syst. **22**(10), 1624–1631 (2011)
5. Chang, R.S., Hu, M.S.: A resource discovery tree using bitmap for grids. Future Gener. Comput. Syst. **26**(1), 29–37 (2010)
6. Qureshi, M.B., Dehnavi, M.M., Min-Allah, N., Qureshi, M.S., Hussain, H., Rentifis, I., Zomaya, A.Y.: Survey on grid resource allocation mechanisms. J.Grid Comput. **12**(2), 399–441 (2014)
7. Cokuslu, D., Hameurlain, A., Erciyes, K.: Grid resource discovery based on centralized and hierarchical architectures. Int. J. Infonomics **3**(1), 227–233 (2010)
8. 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)
9. Brömmel, D., Frings, W., Wylie, B.J.: Extreme-scaling applications en route to exascale. In: ACM Proceedings of the Exascale Applications and Software Conference, p. 1. (2016)
10. Hemamalini, M.: Review on grid task scheduling in distributed heterogeneous environment. Int. J. Comput. Appl. **40**(2), 24–30 (2012)
11. Shalf, J., Dosanjh, S., Morrison, J.: Exascale computing technology challenges. In: International Conference on High Performance Computing for Computational Science. Springer, Berlin, Heidelberg, pp. 1–25 (2010)
12. Hashemi, S.M., Bardsiri, A.K.: Cloud computing vs. grid computing. ARPN J Syst Softw **2**(5), 188–194 (2012)
13. Toporkov, V., Yemelyanov, D., Bobchenkov, A., Potekhin, P.: Preference-based economic scheduling in grid virtual organizations. Procedia Comput. Sci. **80**, 1071–1082 (2016)
14. Guharoy, R., Sur, S., Rakshit, S., Kumar, S., Ahmed, A., Chakborty, S., et al.: A theoretical and detail approach on grid computing a review on grid computing applications. In: IEEE Industrial Automation and Electromechanical Engineering Conference (IEMECON), 2017 8th Annual, pp. 142–146. (2017)
15. Toporkov, V., Toporkova, A., Yemelyanov, D., Bobchenkov, A., Tselishchev, A.: Scheduling optimization in grid with VO stakeholders' preferences. In: International Symposium on Intelligent and Distributed Computing. Springer, Cham, pp. 185–194 (2016)
16. Dawson, C.J., Rick, A.H.I., Joseph, J., Seaman, J.W.: U.S. Patent No. 8,713,179. Washington, DC: U.S. Patent and Trademark Office (2014)
17. Camarinha-Matos, L.M.: Collaborative smart grids—a survey on trends. Renew. Sustain. Energy Rev. **65**, 283–294 (2016)
18. Katyal, M., Mishra, A.: A comparative study of load balancing algorithms in cloud computing environment. (2014). *arXiv preprint* arXiv:1403.6918
19. Allen, Gabrielle, Angulo, David, Foster, Ian, Lanfermann, Gerd, Liu, Chuang, Radke, Thomas, Seidel, Harry, Shalf, John: The cactus worm: experiments with dynamic resource discovery and allocation in a grid environment. IJHPCA **15**, 345–358 (2001). https://doi.org/10.1177/109434200101500402
20. Balazinska, M., Balakrishnan, H., Karger, D.: INS/Twine: A scalable peer-to-peer architecture for intentional resource discovery. In: International Conference on Pervasive Computing, pp. 195–210. Springer, Berlin, Heidelberg (2002)
21. Zuo, X., Iamnitchi, A.: A survey of socially aware peer-to-peer systems. ACM Comput. Surv. CSUR **49**(1), 9 (2016)
22. Olaifa, M., Mapayi, T., Van Der Merwe, R.: Multi ant LA: an adaptive multi agent resource discovery for peer to peer grid systems. In: IEEE Science and Information Conference (SAI), pp. 447–451 (2015).
23. Ranjan, R., Zhao, L., Wu, X., Liu, A., Quiroz, A., Parashar, M.: Peer-to-peer cloud provisioning: service discovery and load-balancing. In: Antonopoulos, N., Gillam, L. (eds.) Cloud computing, pp. 195–217. Springer, London (2010)
24. Noghabi, H.B., Ismail, A.S., Ahmed, A.A., Khodaei, M.: Optimized query forwarding for resource discovery in unstructured peer-to-peer grids. Cybern. Syst. **43**(8), 687–703 (2012)
25. Bazli, B.: Secure, efficient and privacy-aware framework for unstructured peer-to-peer networks (Doctoral dissertation, Liverpool John Moores University) (2016)
26. Jin, X., Chan, S.H.G.: Unstructured peer-to-peer network architectures. In: Handbook of Peer-to-Peer Networking (pp. 117–142). Springer, Boston, MA (2010)
27. Hameurlain, A., Cokuslu, D., Erciyes, K.: Resource discovery in grid systems: a survey. Int. J. Metadata Semant. Ontol. **5**(3), 251–263 (2010)
28. Wu, X., Tavildar, S., Shakkottai, S., Richardson, T., Li, J., Laroia, R., Jovicic, A.: FlashLinQ: a synchronous distributed scheduler for peer-to-peer ad hoc networks. IEEE/ACM Trans. Netw. **21**(4), 1215–1228 (2013)
29. Barbosa, J., Leitão, P., Adam, E., Trentesaux, D.: Dynamic self-organization in holonic multi-agent manufacturing systems: the ADACOR evolution. Comput. Ind. **66**, 99–111 (2015)
30. Prabhakar, B.M., Verma, H.K.: A Comparative study of optimization techniques for optimal reconfiguration of distribution network. Int. J. Adv. Electr. Power Syst. **1**(1), 16–23 (2017)
31. Shi, R., Gan, Y., Wang, Y.: Evaluating scalability bottlenecks by workload extrapolation. In: 2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS) (pp. 333–347). IEEE (2018)
32. Stuardo, C. A., Leesatapornwongsa, T., Suminto, R. O., Ke, H., Lukman, J. F., Chuang, W. C., et al.: Scalecheck: a single-machine approach for discovering scalability bugs in large distributed systems. In: 17th {USENIX} Conference on File and Storage Technologies ({FAST} 19), pp. 359–373 (2019)
33. Singh, S., Chana, I.: QRSF: QoS-aware resource scheduling framework in cloud computing. J. Supercomput. **71**(1), 241–292 (2015)
34. Hameed, A., Khoshkbarforoushha, A., Ranjan, R., Jayaraman, P.P., Kolodziej, J., Balaji, P., Zeadally, S.: A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. Computing **98**(7), 751–774 (2016)
35. Glover, R.: *U.S. Patent No. 9,070,112*. Washington, DC: U.S. Patent and Trademark Office (2015)
36. Ahmed, K., Bigagli, D., Hu, Z., Wang, J.: *U.S. Patent No. 9,632,827*. Washington, DC: U.S. Patent and Trademark Office (2017)
37. Palencia, J.C., Harbour, M.G., Gutiérrez, J.J., Rivas, J.M.: Response-time analysis in hierarchically-scheduled time-partitioned distributed systems. IEEE Trans. Parallel Distrib. Syst. **28**(7), 2017–2030 (2017)
38. Gupta, S., Fritz, C., & De Kleer, J. (2018). *U.S. Patent No. 9,934,071*. Washington, DC: U.S. Patent and Trademark Office
39. Guo, L., Chen, C., Xiaodi, K.E., Jason, T.S.: *U.S. Patent Application No. 15/142,029* (2017)

40. Calo, S.B., Verma, D.C., Bertino, E.: Distributed Intelligence: Trends in the Management of Complex Systems. In: ACM Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies (pp. 1–7) (2017)
41. Zhu, X., Yang, L. T., Jiang, H., Thulasiraman, P., Di Martino, B.: Optimization in distributed information systems (2018)
42. Horelik, N.E.: Domain decomposition for Monte Carlo particle transport simulations of nuclear reactors (Doctoral dissertation, Massachusetts Institute of Technology) (2015)
43. Saurav, S.K., Raghu, H.V., Bapu, S.B.: Self-adaptive power management framework for high performance computing. In: 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI) (pp. 1913–1918). IEEE (2017)
44. Kominar, J.L., Adams, N.P.: *U.S. Patent Application No. 15/152,926* (2017)
45. Orozco, D., Garcia, E., Pavel, R., Khan, R., Gao, G.: TIDeFlow: The time iterated dependency flow execution model. In: 2011 First Workshop on Data-Flow Execution Models for Extreme Scale Computing (pp. 1–9). IEEE (2011)
46. Gong, Q., Zhang, L., Ding, L.: *U.S. Patent No. 9,559,898*. Washington, DC: U.S. Patent and Trademark Office (2017)
47. Sharifi, M., Mirtaheri, S. L., Khaneghah, E. M., & Khaneghah, Z. M. (2011). Process Management Reviewed
48. Khaneghah, E.M.: PMamut: runtime flexible resource management framework in scalable distributed system based on nature of request, demand and supply and federalism." U.S. Patent No. 9,613,312. 4 Apr. 2017
49. Wu, J.: Distributed system design. CRC Press, Boca Raton (2017)
50. Zarrin, J., Aguiar, R.L., Barraca, J.P.: ElCore: dynamic elastic resource management and discovery for future large-scale manycore enabled distributed systems. Microprocess. Microsyst. **46**, 221–239 (2016)
51. Sharifi, M., Khaneghah, E.M., Tirado-Ramos, A., Mirtaheri, S.L.: Formulating the real cost of DSM-inherent dependent parameters in HPC clusters. In: 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), pp. 1–6. IEEE
52. Krekhov, A., Grüninger, J., Schlönvoigt, R., Krüger, J.: Towards in situ visualization of extreme-scale, agent-based, worldwide disease-spreading simulations. In: ACM SIGGRAPH Asia 2015 Visualization in High Performance Computing, p. 7. (2015)
53. Kumar, P., Deokar, S.: Optimal design configuration using HOMER. In: Advances in Systems, Control and Automation, pp. 101–108. Springer, Singapore (2018)
54. Khaneghah, E.M., ShowkatAbad, A.R., Ghahroodi, R.N.: Challenges of Process Migration to Support Distributed Exascale Computing Environment. In: ACM Proceedings of the 2018 7th International Conference on Software and Computer Applications, pp. 20–24.
55. Castain, R.H., Solt, D., Hursey, J., Bouteiller, A.: Pmix: Process management for exascale environments. In: Proceedings of the 24th European MPI Users' Group Meeting (p. 14). ACM (2017)
56. Lefèvre, L., Pierson, J.M.: Introduction to special issue on sustainable computing for ultrascale computing (2018).
57. 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)
58. Cha, M.H., Kim, D.O., Kim, H.Y., Kim, Y.K.: Adaptive metadata rebalance in exascale file system. J. Supercomput. **73**(4), 1337–1359 (2017)
59. Mirtaheri, S.L., Fatemi, S.A., Grandinetti, L.: Adaptive load balancing dashboard in dynamic distributed systems. Supercomput. Front. Innov. **4**(4), 34–49 (2017)
60. 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)
61. Mirtaheri, S. L., Khaneghah, E.M., Sharifi, M.: A case for kernel level implementation of inter process communication mechanisms. In: ICTTA 2008. 3rd International Conference on IEEE Information and Communication Technologies: From Theory to Applications, *2008,* pp. 1–7. (2008)
62. Sharifi, M., Hassani, M., Mousavi, E., Mirtaheri, S.L. (2008, April). Vce: A new personated virtual cluster engine for cluster computing. In: 3rd International Conference on IEEE Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008 (pp. 1–6).
63. Mirtaheri, S.L., Khaneghah, E.M., Grandinetti, L., Sharifi, M.: A mathematical model for empowerment of Beowulf clusters for exascale computing. In: 2013 International Conference on IEEE High Performance Computing and Simulation (HPCS), pp. 682–687 (2013)
64. MM5 (weather model), https://en.wikipedia.org/w/index.php?title=MM5_(weather_model)&oldid=821918188. Last visited 13 Sept 2019
65. Weather research and forecasting model, https://en.wikipedia.org/w/index.php?title=Weather_research_and_forecasting_model&oldid=807407677. Last visited 13 Sept 2019
66. Soltani, N., Khaneghah, E.M., Sharifi, M., Mirtaheri, S.L.: A dynamic popularity-aware load balancing algorithm for structured p2p systems. In: IFIP International Conference on Network and Parallel Computing, pp. 77–84. Springer, Berlin, Heidelberg (2012)
67. Arab, M.N., Mirtaheri, S.L., Khaneghah, E.M., Sharifi, M., Mohammadkhani, M.: Improving learning-based request forwarding in resource discovery through load-awareness. In: International Conference on Data Management in Grid and P2P Systems, pp. 73–82. Springer, Berlin, Heidelberg (2011).
68. Adibi, E., Khaneghah, E.M.: Challenges of resource discovery to support distributed exascale computing environment. Azarbaijan J. High Pefrom. Comput. **1**(2), 168–178 (2018)
69. Khaneghah, E.M., Aliev, A.R., Bakhishoff, U., Adibi, E.: The influence of exascale on resource discovery and defining an indicator.Azarbaijan J. High Pefrom. Comput. **1**(1), 3–19 (2018)

Elham Adibi received master degree from Shahed University and currently is a member of the operating systems and network laboratory at this university. She is interested in High Performance Computing systems and has done some researchers in the mentioned fields. Her studies are detailed in Resource Discovery in high performance computing such as Grid, P2P and Exascale System. From 2015 till now, he is heuristically working on a mathematical mechanism for improving the performance of resource discovery in distributed peer to peer computing systems. She is also interested to design and develop a non-failure resource discovery for distributed 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 an operating system, 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 the 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.