Azerbaijan Journal of High Performance Computing, Vol 1, Issue 1, 2018, pp. 3-19
https://doi.org/10.32010/26166127.2018.1.1.3.19

The Influence of Exascale on Resource Discovery and Defining an Indicator

Ehsan Mousavi Khaneghah[1], Araz R. Aliev[2], Ulphat Bakhishoff[3], Elham Adibi[1]

[1]Department of Computer Engineering, Faculty of Engineering, Shahed University, Tehran, Iran, EMousavi@shahed.ac.ir, elham.adibi@shahed.ac.ir

[2]High Performance Computing Research Advance Center, Department of General and Applied Mathematics, Azerbaijan State Oil and Industry University, Baku, Azerbaijan, alievaraz@asoiu.edu.az

[3] Department of General and Applied Mathematics, Azerbaijan State Oil and Industry University, Baku, Azerbaijan, ulfat.baxıshov@asoiu.edu.az

*Correspondence: Ulphat Bakhishoff, Department of General and Applied Mathematics, Azerbaijan State Oil and Industry University, Baku, Azerbaijan, ulfat.baxıshov@asoiu.edu.az

Abstract

Resource discovery in distributed Exascale computing systems, in addition to managing events resulting in failures due to not finding the resource, as well as failure to perform resource discovery activities at the acceptable time, needs to be able to manage events resulting in failures due to dynamic and interactive nature as well. While investigating the concept of dynamic and interactive nature and its impact on RD functionality, this paper introduces a mathematical model of events resulting in RD failure in this type of computing systems based on descriptive spaces of RD activities. This mathematical model helps to analyze this issue that in what situations the dynamic and interactive nature would lead to the failure of RD and what capabilities RD should have to prevent the failure.

Keyword: Distributed Exascale computing systems, Resource Discovery, Resource discovery failure, Resource discovery indicator.

1. Introduction

Resource Discovery (RD), as one of the units of resource management in computing systems, is responsible for providing the possibility of continuing running a program which requires high performance computing and processing through searching and finding the required processing resource which cannot be answered by the system. [1] RD, unlike other constituent elements of the system manager which operate within the limits of the system, is operating beyond the system and in the system environment. [2] This leads to a reduction in controlling constrain and limitations on activities of RD. The most important consequence of the reduction of these constraints is a lack of a precise view about the nature of events that may occur in running of the RD activities which may affect the process of activities in question in some ways (whether positive or negative). [3, 4]

Lack of a precise understanding of elements related to the RD activities makes the conditions resulting in the failure of RD activities be undebatable when designing the element, unlike other constituent elements of the system manager. [5, 6, 7] About other elements of system manager, since the activities related to the element are executed within the system limits, decisions can be made through analyzing the

function of the element in situations leading to the failure of RD functionality. This decision is based on the knowledge of factors influencing RD activities. In case of RD, since the factors influencing its activities are outside the system limits and uncontrollable for the system manager, therefore, analyzing the effective factors in the process of RD activities, and subsequently, analyzing the conditions resulting in its functions' failure is more complex.

The reason for focusing on the concept of failure of operations in high performance computing and processing systems is due to the functional nature of these types of systems. [11, 12] The main purpose of these systems is to run activities related to the scientific applications in the shortest possible time, [13, 14] hence, running activities related to constituent elements of system manager, from the viewpoint of the functional purpose and its analysis, means increasing the runtime [15]. As a result, when designing the elements forming system manager, mechanisms and patterns will be used which reduce failure in the process of running RD.

The concept of failure of system manager' operations, particularly RD in distributed Exascale systems, has a more complex process due to the definition of the concept of dynamic and interactive nature. [16] Because of the Dynamic and interactive nature, the computing system faces situations that were not considered in the initial structure of answering. [17, 18] On the other hand, the dynamic and interactive nature causes changes in the state of system's basic elements, such as processes, the way of interactions and communications of the basic elements with each other, [19, 20] as well as system limits definition. The nature of these changes is flooding and a change in one part of the system might change the other part. This makes the concept named the difference between the real system state and the perceived system state be definable by an element of system manager. [21]

The occurrence of a dynamic and interactive nature, in many cases, causes the function of system manager, especially RD, to face challenges and fail. From system's functional analysis point of view, failure of RD functionality means conducting an RD process that cannot meet the requirements of the computing processes in the system. [22] Also, after retrieving the modified information on the requirements of the process asking for the resource, RD should re-discover the needed resource.

The above factors make it necessary to have a thorough understanding of the concept of dynamic and interactive nature and its impact on indices describing RD functionality so that decisions can be made on the circumstances leading to failure in RD. [23] While analyzing the concept of dynamic and interactive nature and its impact on RD functionality, this paper also analyzes the concept of failure in RD of distributed Exascale systems. Therefore, it can be discussed what features RD should have to be able to manage the concept of failure in this type of computing systems.

2. Basic concepts

This section examines the basic concepts of distributed Exascale systems and

defines RD functionality in them. [24] The reason is the need for a detailed view of the dynamic and interactive nature and its impact on RD functionality in this type of computing systems. [23] The dynamic and interactive nature can both affect the system's constituent elements and RD functionality. In this paper, it is assumed that RD is not directly influenced by the dynamic and interactive nature and the occurrence of an event in the computing system and influencing the system state will change RD functionality.

2.1. What does Exascale System Functionality mean?

In distributed Exascale systems, all the variables and their relationships, as well as the limits of the computing system are not clear for the designer of the computing system at the beginning of running the program. [25] Therefore, during implementing the program, by linking inter-process communication system processes and creating a new process, a new requirement for a resource is created that has not been taken into account at the system initial design. Such requests, which are completely unpredictable, are due to the dynamic and interactive nature. [26] In distributed Exascale systems, in addition to answering the requirements of the computing processes to continue their activities, RD should also be able to manage the dynamic and interactive nature through providing answering structure. [27]

2.2. What does RD functionality in Distributed Exascale Computing Systems mean?

As one of the units of resource management in computing systems, RD is tasked with receiving a process request that cannot be answered by the local operating system and searching for and finding a resource that is matched with the features and limits of the request outside the computing system. [28] The generating space of RD can be shown as the following equation:

$$\begin{aligned} RD :: \ll & \ Process_{Request}, Resource_{out\ of\ system} >, \\ & < finding, matching, permission/\ allocation, remove\ and\ add \\ & >, Process\ State, \quad Global\ Activity, < Answer, True > > \end{aligned}$$

Eq.1

As can be seen in Eq.1, RD is defined based on $Process_{Request}$ and $Resource_{out\ of\ system}$ sets. These are the two main sets which define RD. In fact, RD is a function responsible for mapping $Resource_{out\ of\ system}$ to $Process_{Request}$. To perform this mapping, RD should use four activities.

In the first activity, RD should find the resource out of the computing system based on any mechanisms.

For the second activity, according to the system manager policy, RD can provide the process with either a resource which is 100% matched with the process request or a resource with a minimum specific resemblance determined by system manager policy. In traditional computing systems, such as grid and peer-to-peer, RD uses 100% match policy. [29] The reason is the clarity of answering structure. In distributed Exascale systems, the clarity of answering structure as well as time limits can minimally be resembled. [30, 31].

RD should have access rights to the resource. These access rights can be either total or partial. Access rights can also be a function of time or non-compliance with

time. In total access, RD provides the process with the discovered resource as part of the local resources of the requesting process. In this case, system manager is responsible for establishing the transparency. In partial access, RD only allows specific activities for the requesting process, and the requesting process must establish and manage a distinct activity (part of a global activity) to run the activity in the remote resource. [32]

Based on the general policies of system manager, RD should make decisions on permanent or temporary addition of the computing element containing the requested resource. If the global activity has a high operating frequency or system manager operates based on the concept of global activity similarity, then RD will typically behave according to the concept of permanent addition, and in other cases, based on process needs and requests. The concept of global activity similarity is used in computing systems with low frequency of running a global activity, but global activities carried out by RD are similar to each other. [33] In this situation, system manager uses a management pattern of the global activity that has the most similarity in terms of the implementation process, the beneficiary elements, as well as the activity requests to manage a global activity.

On the other hand, RD should make decisions on reducing system size and removing unused machines in the system based on system manager policies. One of the challenges of distributed Exascale systems, as compared to open computing systems, is the presence of unused machines in the system which increases the runtime of constituent elements of system manager. In this type of computing systems, RD should decide on removing unused machines. [34]

Two basic concepts which define RD activities are process state and global activity. The concept of process state involves the causes of creating and the nature of the request in the process, and global activity state reflects having or lacking memory of the request process. [23, 35]

Each activity executed by RD must have attributes of Answer and True. Answer means whether RD is capable of answering a process request or not.

RD, contrary to load balancer, operates beyond the system limits. The computing elements outside the computing system lack the necessary constraints to participate in answering process. On the other hand, these elements do not follow the rules governing the computing system. Therefore, in the answering process, RD may encounter a concept known as inability to answer. Based on a criterion, RD should make decisions on whether it is capable of answering the process request or not. [36]

Given the definition of the reference system, it can be argued that RD is able to answer any request in an infinite time and location. On the other hand, in computing systems, including open computing systems or distributed Exascale systems, a process creates a request. Each process has a concept called receiving a request in the appropriate time and location. [12] If the appropriate time and location related to the process request is violated, the trend of process life may face problems. As a result, RD Answer means a significant answer to process request. It is more complex in distributed Exascale systems. In this type of computing systems, the occurrence of

dynamic and interactive events during resource discovery process may change significance of answering to the request; hence, in such systems, RD requires using more precise mechanisms to be Answer. [37]

The concept of True points to the fact that the request must be answered correctly. Correct answer to a request means lack of failure of RD during resource discovery process.

In this type of computing systems, scalability, and subsequently, RD functionality is not limited to creating answer structure or adding (or removing) computing elements to the computing system. Here, RD is also used to deal with the dynamic and interactive nature. The dynamic and interactive nature leads to the creation of a new request or changing the constraints and limitations of the current request. In addition to receiving the requests of processes that need a resource, RD has also the task of analyzing the request state and the requesting process by some mechanisms. Moreover, in case of changes in resources and process attributes over time, RD should be able to manage the situation in a way that does not lead to the stoppage of the current resource and the start of discovering a new resource, and in other words, should prevent RD failure. [38]

RD in distributed Exascale systems receives a process request in terms of the request nature, time and location constraints, and type of requested resource, as well as the access constraints defined therein. Because of the possibility of changes in any of the mentioned constraints, the activity related to resource discovery becomes more complicated. In addition, since the activity related to this element is beyond the limits of the computing system and the elements do not have any constraints in giving services to the end of implementing, the likelihood of resource discovery failure increases. Providing mechanisms and methods to control and manage it is of great importance in increasing the computing system efficiency.

3. Related work

Aiming at sharing audio files, Napster is one of the first peer-to-peer systems. In this system, the central management element is intended to carry out the processes related to finding the resource and establishing the necessary link between the elements requesting the resource and those containing the resource. Each computing element provides central management element with the information on resources that are able to share them with other computing elements. [39, 40] If the information of the computing element resources changes during the execution of the program and the updated information is not presented to central management element, it will redirect query to a wrong computing element and the activity related to the resource will fail. Furthermore, if central management element has some defects for any reasons, such as hardware or software, it will be unable to direct the request and process the query, and thus, executing resource discovery activities will be stopped. [41]

One of the resource finding mechanisms used in the Gnutella resource sharing system is flooding. In this method, a processor in the system requests a resource which local operating system is not able to answer it. In this case, the resource

request will be spread to all the neighbors of the requesting process. If none of the neighbors is capable of answering the request, they will send the resource request to their neighbors. This method can exponentially increase the network overload. [42, 43]

In order to prevent too many queries, an indicator called time to live is used. TTL is defined in the resource requesting process, indicating that to how many computing elements, at most, the request message can be spread. When passing through any computing element, one unit of this indicator value will be reduced. If the TTL value reaches zero, the activity associated with the resource discovery will be terminated. Therefore, a failure occurs due to not finding a resource. [44]

Due to system's scalability, computing systems running scientific applications have a dynamic nature. That is, each computing element can be deleted from or added to the system and its resource state can be changed. [45]

Paper [46] has proposed a model for establishing a relationship between RD and load balancer. To efficiently perform resource discovery operations, these two elements send or receive the necessary information by exchanging command messages. Load balancer has sufficient knowledge and awareness of resource states of each of the system's computing elements and knows how much of the load work of each of them is being used and will be used in the future. If the load balancer provides its information to RD, RD will be able to make appropriate decisions regarding the changing state of computing resources. Therefore, by the indicator of calculating and predicting computing elements' workload, this paper prevents resource discovery failure from the presence or absence of computing elements and their inability to answer the resource request.

4. Failure-Aware ExaRD

RD, from its systemic point of view, can be considered as a global activity aimed at finding a resource that is requested by a process in the computing system that cannot be answered in a local computing system. [47] In traditional computing systems, such as Grid and Peer to Peer, RD function was defined based on the independent variable of resource request and the conditions governing the request, and the dependent variable of finding (or not finding) the resource. The central element in defining such a function is based on the concept of resource type. [48] However, in Exascale systems, the concept of dynamic and interactive nature directly affect RD functionality and its influencing factors. RD functionality is dependent on Resource Nature Set and Resource Attribute Set. Beta Resource Nature Set can be written as Eq.2.

$$\mathrm{RNS}_{\mathrm{Alpha}}(\mathrm{Beta}) :: < \mathrm{Request}_{\mathrm{Nature}}, \mathrm{Request}_{\mathrm{type}}, \mathrm{Request}_{\mathrm{time}}, \mathrm{Request}_{\mathrm{location}}, \mathrm{RAC}_{\mathrm{Beta}}, \mathrm{Permission}, \mathrm{Allocation} >$$

Eq. 2

As seen in Eq.2, RNA set of Beta request is defined on the four spaces of request nature, request type, request time limit, and request location limit. In this series, permission and resource allocation to the process can be defined.

In traditional systems such as Grid and Peer to Peer, RNS does not change

during implementing activities related to RD. [48] On the other hand, RD in this type of computing systems tries to provide the computing element with a process that its RAS set will not change during answering the process request.

Any changes in RNS set and the high frequency of changes in RAS set will create a concept called the failure of RD related activities. In distributed Exascale computing systems, depending on the dynamic and interactive nature of computing processes, RNS set of this type of computing processes may change during answering to a process request or resource discovery. On the other hand, in this type of computing systems, depending on the distributed nature of the system and the definition of local autonomy [49], the frequency of RAS set changes may increase.

When a request is formed in the system, regardless of its cause, and in order to answer it by load balancer or RD, each request is considered as <time, type, location> from their perspective. When Beta request is created in alpha process, either RD or load balancer will be called. At the moment of calling, each of these two elements considers the triple schema as "Request Image". In traditional computing systems, Request Image is constant during answering by either of these two elements. In distributed Exascale systems, due to the events from the dynamic and interactive nature of such systems, it is possible to make changes in each of the constituent elements of RI.

Therefore, the dynamic and interactive nature forms Beta request in the system where its RNS differs from the initial RNS of the global activity, or the Request Image from it changes during the implementation process of RD or load balancer. Thus, distributed Exascale systems, needs to have the capability of examining the constraints governing the request during the program implementation.

Based on the information related to the process state and analysis of process request state, if a resource discovery mechanism is able to

A) If URS or the User Request Set is a set of process requirements that cannot be answered in the local machine in the local computing system and is in the form of

<RAC, Limitation time, Limitation Location>,

Then, when examining the URS of a computing element, it must be able to re-examine the URS with the resource requesting process or the representative resource requesting element. It is called double verification RNS.

B) It should be able to examine the stated condition in Eq.3 when checking URS of a computing element.

$$if\ URS \equiv RNS \overset{To\ somehow\ that}{\therefore} RI(t)\ is\ constant$$

Eq.3

Eq.3 states that if the URS of the computing element is capable of meeting the RNS of the requesting process, RD can add the new computing element to the computing system when it does not change system's RI state at the time of discovery. If adding the new computing element changes RI state of the computing system, due to the concept of change chain, even if the URS of the computing element is able to meet RNS, this computing element will be disregarded.

t = zeta is the start of RD operation and t = zeta2 is its completion. During [zeta,

zeta2] time slot, due to the occurrence of a dynamic and interactive nature, the computing system is subject to some changes and is considered as a dynamic system. In terms of changes to the beneficiary elements, system's dynamism is important for resource requesting process or activities related to RD. Because the system is in a dynamic state, RNS is changing. If RNS value at t = zeta is equal to RNSzeta, RD must find a resource at t = zeta2 that while considering RNS changes from RNSzeta to RNSzeta2, will not change RI at zeta and zeta2.

Regarding these concepts, it can be argued that the dynamic and interactive nature influences distributed Exascale systems in two areas: a) a change in request nature after activating RD; b) a change in RI state after activating RD. It has been discussed in Eq.4.

$$ExaRD: (Dynamic\ and\ Interactive) \rightarrow \left(Change_{RNS}(t), RI(t)\right)$$

Eq.4

As shown in Eq.4, the dynamic and interactive nature affects RD in two areas of RNS time changes due to the occurrence of a dynamic and interactive nature on the requesting process or its representative, or system's RI state change which somehow influences the process request. The concept of RI (t) focuses on two concepts of system state and the creation of change chain. From the system's approach, RI is a function of the independent variable t and its state may change because of the dynamic and interactive nature either by the requesting process or by other processes affecting requesting process function. From the change chain approach, the discovery of a resource whose URS is the same as RNS request should not cause the formation of a chain of requirements or a new requirement in the system. Given Eq.4, it can be stated that if RD algorithms used by RD can support two attributes of function 4 range, then, they can be used in distributed Exascale systems.

Eq.4 is, in fact, the use of a mapping function to analyze the effect of the dynamic and interactive nature on RD. Figure.1 displays the schematic model of Eq.4.

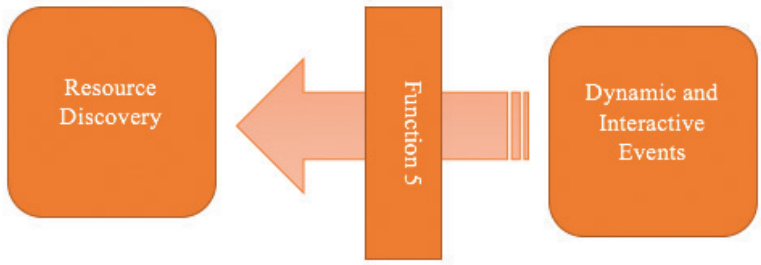

Figure.1. Schematic model of Eq.4.

As seen in Fig.1, the concept of dynamic and interactive nature affects RD

functionality. As stated in Eq.4, the effect of this concept on RD functionality causes the two concepts of RI and RNS to be influences by time. In traditional computing systems, from the moment of RD activation until answering the request by RD, these two concepts are constant and do not change.

Given the aforementioned issues, it can be stated that RD is made up of RI, RNS, URS, and RAC spaces. The formation of dynamic and interactive nature makes it possible for two spaces to change from time-independent to time-dependent spaces. As a result, all four spaces are converted into time-dependent variables. If it is assumed that $RD = \begin{bmatrix} RNS & URS \\ RAC & RI \end{bmatrix}$, then if we calculate the determinant of this matrix according to Eq.6,

$$\lfloor RD \rfloor = (RNS{*}RI)\text{-}(URS{*}RAC)$$

Eq.6

the matrix determinant will be a 2-line function that maps and describes RD state for an entry constancy. Consequently, Eq.5 allows the computing system designer to examine the impact of the three constituent spaces of RD when an element is in a constant state. Eq.5 can be described in such a way that if we assume that the main function of RD is to find and allocate resources, then, in RD matrix, we will seek to find a 2-line function that shows the effect of the constituent elements of RD when RAS is constant. In traditional computing systems, this function always has a fixed and constant value. The cause of ⌊RD⌋ constancy is that RNS, RI, URS, and RAC spaces are not dependent on the changeable independent variable. However, in distributed Exascale systems, RI and RNS are functions of time; thus, ⌊RD⌋ is a function of the independent variable of time. Then, Eq.6 will be as follows:

$\lfloor RD \rfloor_t = [(RNS{*}RI)\text{-}(URS{*}RAC)]_t$

$$\lfloor RD \rfloor_t = \begin{bmatrix} [Constant & < Time\ Limitation & Location\ Limitation >]. \\ & \begin{bmatrix} Request_{Nature} \\ Request_{type} \end{bmatrix} & \end{bmatrix}_t$$

Eq.6

As shown in Eq.6, the function of RD which is based on the four spaces of RAC, RI, URS, and RAC and considering time as the independent variable, can be obtained by calculating the determinant of RD generating matrix. This function is a matrix function and is defined in the form of inner product of 1*2 matrix by 2*1 matrix. The first matrix is referred to as limitation matrix and the second one as request matrix. In the first matrix, constant is any kind of limitation that is defined on RD process by resource requesting process or the computing system, except for time and location limitations. In this matrix, time and location limitations are those imposed by resource requesting process or the computing system on RD in an RD process. In the second matrix, the request nature and type are described. Each of the four parameters are considered as functions of the independent variable of time and can be changed during the runtime of RD activity.

In order to have a more accurate view on the impact of the above concept on RD functionality, we need to define RD. Eq.1 shows the generating spaces of RD. Given Eq.1, RD can be defined as follows:

$$\mathrm{RD} = \frac{\partial \begin{bmatrix} Process_{State} & Global_{Activity} \\ <True,Answer> & Resource_{Nature} \end{bmatrix}}{\partial_{(resource,request,time,location)}}$$

Eq.7

Eq.7 suggests that RD is an element with the task of answering a process request process at the right time and location. It has to answer an appropriate request which is in accordance with the requirements of the requesting process at the right time and location by taking into account the constraints and limitations defined by the process or system. Therefore, in Eq.7, the elements influencing RD concept are derived from the two main elements of RD and time and location limitations. By solving Eq.7, Eq.8 will be obtained.

$$RD = \begin{bmatrix} \frac{\partial Process_{State}}{\partial resource} & \frac{\partial Process_{State}}{\partial request} & \frac{\partial Process_{State}}{\partial time} & \frac{\partial Process_{State}}{\partial location} \\ \frac{\partial Global_{Activity}}{\partial resource} & \frac{\partial Global_{Activity}}{\partial request} & \frac{\partial Global_{Activity}}{\partial time} & \frac{\partial Global_{Activity}}{\partial location} \\ \frac{\partial < True, Answer >}{\partial resource} & \frac{\partial < True, Answer >}{\partial request} & \frac{\partial < True, Answer >}{time} & \frac{\partial < True, Answe}{\partial location} \\ \frac{\partial Resource_{Nature}}{\partial resource} & \frac{\partial Resource_{Nature}}{\partial request} & \frac{\partial Resource_{Nature}}{\partial time} & \frac{\partial Resource_{Natu}}{\partial location} \end{bmatrix}$$

Eq.8

As can be seen in Eq.8, the derivative of the RD-generating matrix on four main concepts of RD defines RD. RD described in Eq.8 should be equivalent to RD in Eq.6. Therefore:

$$\left[[Constant \quad < Time\ Limitation \quad Location\ Limitation >]. \begin{bmatrix} Request_{Nature} \\ Request_{type} \end{bmatrix} \right]_t$$

$$= \begin{bmatrix} \frac{\partial Process_{State}}{\partial resource} & \frac{\partial Process_{State}}{\partial request} & \frac{\partial Process_{State}}{\partial time} & \frac{\partial Process_{State}}{\partial location} \\ \frac{\partial Global_{Activity}}{\partial resource} & \frac{\partial Global_{Activity}}{\partial request} & \frac{\partial Global_{Activity}}{\partial time} & \frac{\partial Global_{Activity}}{\partial location} \\ \frac{\partial < True, Answer >}{\partial resource} & \frac{\partial < True, Answer >}{\partial request} & \frac{\partial < True, Answer >}{time} & \frac{\partial < True, Answe}{\partial location} \\ \frac{\partial Resource_{Nature}}{\partial resource} & \frac{\partial Resource_{Nature}}{\partial request} & \frac{\partial Resource_{Nature}}{\partial time} & \frac{\partial Resource_{Natu}}{\partial location} \end{bmatrix}$$

Eq.9

By differentiating from the left matrix, Eq.8 will be obtained according to the

independent variable of time.

$$\frac{\partial Request_{nature}}{\partial time} + location\ limitation + \frac{\partial location\ limitation}{\partial time} * time\ limiation * \frac{\partial Request_{type}}{\partial time} = \begin{bmatrix} \frac{\partial Process_{State}}{\partial resource} & \frac{\partial Process_{State}}{\partial request} & \frac{\partial Process_{State}}{\partial time} & \frac{\partial Process_{State}}{\partial location} \\ \frac{\partial Global_{Activity}}{\partial resource} & \frac{\partial Global_{Activity}}{\partial request} & \frac{\partial Global_{Activity}}{\partial time} & \frac{\partial Global_{Activity}}{\partial location} \\ \frac{\partial < True, Answer >}{\partial resource} & \frac{\partial < True, Answer >}{\partial request} & \frac{\partial < True, Answer >}{time} & \frac{\partial < True, Answe}{\partial location} \\ \frac{\partial Resource_{\mathrm{Nature}}}{\partial resource} & \frac{\partial Resource_{Nature}}{\partial request} & \frac{\partial Resource_{Nature}}{\partial time} & \frac{\partial Resource_{Natu}}{\partial location} \end{bmatrix}$$

Eq.10

By solving Eq.10, it can be concluded that to consider time limitations associated with RNS and converting RI from a time-dependent function into a time-dependent function, it should be able to take into account $\frac{\partial Request_{nature}}{\partial time}$, $\frac{\partial Request_{type}}{\partial time}$, $\frac{\partial Global_{Activity}}{\partial resource,location}$, and $\frac{\partial Global_{Activity}}{\partial time}$.

Based on Eq.4 and Fig.1, RD can be used in distributed Exascale systems if it can consider the four mentioned variables. Eq.4 emphasizes that RD can be used in distributed Exascale systems when:

A) During RD process, RD gets information about RI changes in some way. In traditional computing systems, an RI is created at the moment of calling and starting RD operation, based on three concepts of request and time and location limitations and the main assumption of RD is not changing the created RI during RD process. However, in distributed Exascale computing systems, RI may change during RD process due to the occurrence of a dynamic and interactive nature of resource requesting process or any other process that somehow influences it. From the 4 mentioned variables, $\frac{\partial Request_{type}}{\partial time}$ and $\frac{\partial Global_{Activity}}{\partial resource,location}$ will be used to convert RI from a time-independent variable into a time-dependent variable.

B) RD somehow gets information on RNS changes during RD process. Regardless of the space for defining RNS variable and the complexities of defining RNS space, as well as the pattern of defining RNS space, it can be suggested that this space was created to define the request nature. In traditional computing systems, the nature of the request that activates RD does not change during running RD activities. However, in distributed Exascale systems, the occurrence of a dynamic and interactive nature may lead to a change in request nature. Among the four mentioned variables, $\frac{\partial Request_{nature}}{\partial time}$ and $\frac{\partial Global_{Activity}}{\partial time}$ will be used to get information on RNS variable during RD process. Therefore, RD can be used in distributed Exascale systems if conditions (a) to (c) are met.

A) Variable $\frac{\partial Request_{nature}}{\partial time}$ means that RD should be able to consider changes in the request nature within time. This variable should be able to provide RD with the capability of considering RNA changes during RD. Regardless of the pattern of RNS space definition, the most important reason for creating this space is to define the request nature. As a result, $\frac{\partial Request_{nature}}{\partial time}$ tries to describe changes in the request nature or RNS over time. If RD wants to measure $\frac{\partial Request_{nature}}{\partial time}$, it must be able to gets information on the state of request changes during RD based on a mechanism. The nature of $\frac{\partial Request_{nature}}{\partial time}$ is of partial derivative. Hence, in the ideal state, it should be calculated at any moment of RD. On the other hand, RNS changes when the process activating RD creates a dynamic and interactive nature in its interactions with other elements of global activity. Although the dynamic and interactive nature and its occurrence are such that cannot be accurately predicted, but given that the operations running in the computing systems take advantage of replication model, decisions need to be made on the time of partial derivative calculation in order to examine whether RNS is related to the request that has led to global activity or not.

RD uses Eq.11 to decide on RNS change.

$$\forall i, j \in Run_{Time} :: \frac{\partial Request_{nature}}{\partial time}\ (i, \beta) \neq \frac{\partial Request_{nature}}{\partial time}\ (j, \beta) => RD\ is\ Reset$$

Eq.11

Eq.11 states that if the value of $\frac{\partial Request_{nature}}{\partial time}$ of β process changes in i and j moments, then, RD must refer again to the process activating RD to discover a resource by using a new RNS, in the event of a change in RNS. Calculating $\frac{\partial Request_{nature}}{\partial time}$ means RD manger should be able to obtain information on the request nature by using data structure of the operating system.

B) $\frac{\partial Global_{Activity}}{\partial time}$ means that RD should be able to consider the process state changes in the global activity of the time ratio. Otherwise, any other changes in the functional and behavioral nature of the process RD must be answered by a local operating system. RD uses $\frac{\partial Global_{Activity}}{\partial time}$ to calculate the relative time estimate in order to calculate the variable mentioned in (A). $\frac{\partial Global_{Activity}}{\partial time}$ is a partial derivative and its interpretation implies that at what times the process role in the global activity will change. Typically, when the process role in the global activity changes, a dynamic and interactive nature has occurred. The occurrence of a dynamic and interactive nature can lead to a change in the request nature; as a result, $\frac{\partial Request_{nature}}{\partial time}$ have to be calculated so that decisions can be made on RNS changes in the system. Using the concept of the computing activity repeatability, the history of running the activity, the model governing the global activity, and calculating $\frac{\partial Global_{Activity}}{\partial time}$, RD should be able to make decisions on times when calculating the variable is following the model suggested in Eq.12.

$$\forall i,j \in Run\ Time :: \frac{\partial Global_{Activity}}{\partial time}(i) \neq \frac{\partial Global_{Activity}}{\partial time}(j)$$

Eq.12

Eq.12 implies that the process state causing the activation of RD, must be different at two different moments of i and j. Differing states of the process mentioned in i and j means that the interactions between processes forming the global activity may lead to a dynamic and interactive nature, and consequently, may change RNS. After reaching the stable state, the frequency of calculating $\frac{\partial Global_{Activity}}{\partial time}$ can be considered as the calculating times of the variable in (A).

C) $\frac{\partial Request_{type}}{\partial time}$ means calculating changes in request type relative to time changes. RI variable indicates the concepts of request and time and location limitations. Calculating $\frac{\partial Request_{type}}{\partial time}$ implies that whether RI has changed or not. If $\frac{\partial Request_{type}}{\partial time}$ changes, RD will interpret it as a change in one of the constituent elements of RI, mostly request change. Like Eq.11, RD can calculate $\frac{\partial Request_{type}}{\partial time}$ at i and j, and if there is a change in the variable, it will return to the process activating RD to receive the new RI state. As stated in (A), since $\frac{\partial Request_{type}}{\partial time}$ is a partial derivative variable, as a result, one of the main challenges of $\frac{\partial Request_{type}}{\partial time}$ is the time and frequency of calculating this variable.

$\frac{\partial Global_{Activity}}{\partial resource,location}$ indicates of the frequency of changes in the process activating global activity in relation to the resource and location limitations governing the global activity. As discussed about variable (b), the process activating RD, will change due to interactions with other elements of global activity, which can lead to a dynamic and interactive nature, and consequently, RI change. Changes in location limitations of the global activity and resources in the global activity will change the RI state of the process, thus, this change should be considered in relation to the global activity state. As stated in (B), using the concept of history, the governing model of the global activity, as well as the frequency of global activity changes, RD decides on times to calculate this variable. The frequency obtained for this variable can be regarded as the frequency of calculating the variable shown in (C).

5. Conclusion

RD can be used in distributed Exascale systems when: (a) it can analyze the functional and behavioral state of the process in the global activity. This means that RD should not consider the computing process as an abstract element, rather, should be able to consider it as part of a global activity. (b) It has mechanisms and methods to analyze changes in the process request nature during RD process. This implies that it should either be directly in connection with the process requesting and activating RD or can analyze the effects of other elements that change the request nature. It should also be able to get information based on the mechanism of

checking the process behavior in relation to the frequency of request nature changes. It means having information on the requests' nature and their changing time. (c) It has mechanisms and methods for analyzing the requests' type. Because of this, RD should have a pattern for classifying resources as well as answerable resources and make decisions on changing the request type based on this pattern. To do such, RD should either be directly in connection with the process requesting and activating RD or can make decisions on changing the request type by examining the impacts of factors affecting the process. It should also be able to decide on the frequency of changing the process request type by time check of the process.

Reference

[1]. Qureshi, M. B., Dehnavi, M. M., Min-Allah, N., Qureshi, M. S., Hussain, H., Rentifis, I., & Zomaya, A. Y. (2014). Survey on grid resource allocation mechanisms. *Journal of Grid Computing,* 12(2), 399-441.

[2]. Black, B., Roersma, J. S., Boelens, J., Dunbar, N., Lange, S., & Swanson, W. (2014). U.S. Patent No. 8,856,329. Washington, DC: U.S. Patent and Trademark Office.

[3]. Souri, A., & Navimipour, N. J. (2014). Behavioral modeling and formal verification of a resource discovery approach in Grid computing. *Expert Systems with Applications,* 41(8), 3831-3849.

[4]. Samimi, P., Teimouri, Y., & Mukhtar, M. (2016). A combinatorial double auction resource allocation model in cloud computing. *Information Sciences, 357,* 201-216.

[5]. Messina, F., Pappalardo, G., Rosaci, D., Santoro, C., & Sarné, G. M. (2013). A trust-based approach for a competitive cloud/grid computing scenario. *In Intelligent Distributed Computing VI* (pp. 129-138). Springer, Berlin, Heidelberg.

[6]. Zhong, H., Tao, K., & Zhang, X. (2010, July). An approach to optimized resource scheduling algorithm for open-source cloud systems. *In ChinaGrid Conference (ChinaGrid), 2010 Fifth Annual* (pp. 124-129). IEEE.

[7]. Banerjee, S., & Hecker, J. P. (2017). A Multi-agent system approach to load-balancing and resource allocation for distributed computing. In *First Complex Systems Digital Campus World E-Conference 2015* (pp. 41-54). Springer, Cham.

[8]. Mahmud, R., Kotagiri, R., & Buyya, R. (2018). Fog computing: A taxonomy, survey and future directions. *In Internet of everything* (pp. 103-130). Springer, Singapore.

[9]. Navimipour, N. J., & Milani, F. S. (2015). A comprehensive study of the resource discovery techniques in peer-to-peer networks. *Peer-to-Peer Networking and Applications, 8(3),* 474-492.

[10]. Wale, N., Sim, D. G., Jones, M. J., Salathe, R., Day, T., & Read, A. F. (2017). Resource limitation prevents the emergence of drug resistance by intensifying within-host competition. *Proceedings of the National Academy of Sciences,* 114(52), 13774-13779.

[11]. Wang, D., He, D., Wang, P., & Chu, C. H. (2015). Anonymous two-factor authentication in distributed systems: certain goals are beyond attainment. *IEEE Transactions on Dependable and Secure Computing, (1),* 1-1.

[12]. Wu, J. (2017). ***Distributed system design.*** CRC press.

[13]. Balakrishnan, B., Kothamasu, V. R., & Woods, G. (2015). *U.S. Patent No. 9,032,369.* Washington, DC: U.S. Patent and Trademark Office.

[14]. Li, K., Tang, X., Veeravalli, B., & Li, K. (2015). Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems. *IEEE Transactions on computers,* 64(1), 191-204.

[15]. Bhalachandra, S., Porterfield, A., & Prins, J. F. (2015, May). Using dynamic duty cycle modulation to improve energy efficiency in high performance computing. *In Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International* (pp. 911-918). IEEE.

[16]. Khaneghah, E. M., ShowkatAbad, A. R., & Ghahroodi, R. N. (2018, February). Challenges of Process Migration to Support Distributed Exascale Computing Environment. *In Proceedings of the 2018 7th International Conference on Software and Computer Applications* (pp. 20-24). ACM.

[17]. Babuji, Y. N., Chard, K., Gerow, A., & Duede, E. (2016, October). A secure data enclave and analytics platform for social scientists. *In e-Science (e-Science), 2016 IEEE 12th International Conference on* (pp. 337-342). IEEE.

[18]. Totu, L. C., Leth, J., & Wisniewski, R. (2013, June). Control for large scale demand response of thermostatic loads. *In ACC (pp. 5023-5028).*

[19]. Sadeghi, A. R., Wachsmann, C., & Waidner, M. (2015, June). Security and privacy challenges in industrial internet of things. *In Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE (pp. 1-6)*. IEEE.

[20]. Kilian, F., & Luik, O. (2013). U.S. Patent No. 8,533,717. Washington, DC: U.S. Patent and Trademark Office.

[21]. Xavier, M. G., Neves, M. V., Rossi, F. D., Ferreto, T. C., Lange, T., & De Rose, C. A. (2013, February). Performance evaluation of container-based virtualization for high performance computing environments. *In Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on* (pp. 233-240). IEEE.

[22]. Sakadasariya Achyut, R. Survey of Resource and Job Management for Load Bal-ancing In Grid Computing. *Of the IJISME ISSN,* 2319-6386.

[23]. Khaneghah, E. M. (2017). *U.S. Patent No. 9,613,312.* Washington, DC: U.S. Patent and Trademark Office.

[24]. Mousavi Khaneghah, E., Noorabad Ghahroodi, R., & Reyhani ShowkatAbad, A. (2018). A mathematical multi-dimensional mechanism to improve process migration efficiency in peer-to-peer computing environments. *Cogent Engineering, 5(1),* 1458434.

[25]. Reed, D. A., & Dongarra, J. (2015). Exascale computing and big data. *Communications of the ACM, 58(7),* 56-68.

[26]. Wang, K., Kulkarni, A., Lang, M., Arnold, D., & Raicu, I. (2016). Exploring the design tradeoffs for extreme-scale high-performance computing system software. *IEEE Transactions on Parallel and Distributed Systems, 27(4),* 1070-1084.

[27]. Wang, K., Zhou, X., Li, T., Zhao, D., Lang, M., & Raicu, I. (2014, October). Optimizing load balancing and data-locality with data-aware scheduling. *In Big Data (Big Data), 2014 IEEE International Conference on (pp. 119-128).* IEEE.

[28]. Towns, J., Cockerill, T., Dahan, M., Foster, I., Gaither, K., Grimshaw, A., & Roskies, R. (2014). XSEDE: accelerating scientific discovery. *Computing in Science & Engineering,* 16(5), 62-74.

[29]. Zhu, X., Yang, L. T., Jiang, H., Thulasiraman, P., & Di Martino, B. (2018). Optimization in distributed information systems.

[30]. Horelik, N. E. (2015). *Domain decomposition for Monte Carlo particle transport simulations of nuclear reactors* (Doctoral dissertation, Massachusetts Institute of Technology).

[31]. Saurav, S. K., Raghu, H. V., & Bapu, S. B. (2017, September). Self-adaptive power management framework for high performance computing. *In Advances in Computing, Communications and Informatics (ICACCI), 2017 International Conference on* (pp. 1913-1918). IEEE.

[32]. Kominar, J. L., & Adams, N. P. (2017). *U.S. Patent Application No.* 15/152,926.

[33]. Orozco, D., Garcia, E., Pavel, R., Khan, R., & Gao, G. (2011, October). 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.

[34]. Gong, Q., Zhang, L., & Ding, L. (2017). *U.S. Patent No. 9,559,898.* Washington, DC: U.S. Patent and Trademark Office.

[35]. Sharifi, M., Mirtaheri, S. L., Khaneghah, E. M., & Khaneghah, Z. M. (2011). Process Management Reviewed.

[36]. Navimipour, N. J., Rahmani, A. M., Navin, A. H., & Hosseinzadeh, M. (2014). Resource discovery mechanisms in grid systems: A survey. *Journal of Network and Computer Applications,* 41, 389-410

[37]. Zarrin, J., Aguiar, R. L., & Barraca, J. P. (2016). ElCore: Dynamic elastic resource management and discovery for future large-scale manycore enabled distributed systems. *Microprocessors and Microsystems, 46,* 221-239.

[38]. Sambasivan, R. R., Zheng, A. X., De Rosa, M., Krevat, E., Whitman, S., Stroucken, M., & Ganger, G. R. (2011, March). Diagnosing Performance Changes by Comparing Request Flows. In NSDI (Vol. 5, pp. 1-1).

[39]. Gopal, S. V., Rao, N. S., & Naik, S. L. (2016, March). Dynamic sharing of files from disconnected nodes in peer to peer systems. *In Electrical, Electronics, and Optimization Techniques (ICEEOT), International Conference on* (pp. 767-770). IEEE.

[40]. Rodrigues, R., & Druschel, P. (2010). Peer-to-peer systems. *Communications of the ACM,* 53(10), 72-82.

[41]. Selvaraj, C., & Anand, S. (2012). A survey on security issues of reputation management systems for peer-to-peer networks. *Computer Science Review,* 6(4), 145-160.

[42]. Bandara, H. D., & Jayasumana, A. P. (2013). Collaborative applications over peer-to-peer systems–challenges and solutions. *Peer-to-Peer Networking and Applications, 6(3),* 257-276.

[43]. Asghari, S., & Navimipour, N. J. (2018). Resource discovery in the peer to peer networks using an inverted ant colony optimization algorithm. *Peer-to-Peer Networking and Applications,* 1-14.

[44]. Palmieri, F. (2017). Bayesian resource discovery in infrastructure-less networks. *Information Sciences,* 376, 95-109.

[45]. Jiang, C., Gao, L., Duan, L., & Huang, J. (2018). Scalable mobile crowdsensing via peer-to-peer data sharing. *IEEE Transactions on Mobile Computing,* 17(4), 898-912.

[46]. Arab, M. N., & Sharifi, M. (2014). A model for communication between resource discovery and load balancing units in computing environments. *The Journal of Supercomputing,* 68(3), 1538-1555.

[47]. Thomas, D., Baron, J., & Raymond, M. A. (2015). *U.S. Patent Application* No. 13/930,955.

[48]. Hussain, H., Malik, S. U. R., Hameed, A., Khan, S. U., Bickler, G., Min-Allah, N., & Kolodziej, J. (2013). A survey on resource allocation in high performance distributed computing systems. *Parallel Computing, 39(11),* 709-736.

[49]. Yu, W., Liu, D., & Yu, N. (2013). Feeder control error and its application in coordinate control of active distribution network [J]. *Proceedings of the CSEE,* 33(13), 108-115.

Submitted 01.02.2018
Accepted 02.05.2018