Azerbaijan Journal of High Performance Computing, Vol 4, Issue 2, 2021, pp. 170-187
https://doi.org/10.32010/26166127.2021.4.2.170.187

ExaLazy: A Model for Lazy-Copy Migration Mechanism to Support Distributed Exascale System

Ehsan Mousavi Khaneghah[1], Tayebeh Khoshrooynemati[1], Azar Feyziyev[2]

[1] *Shahed University, Tehran, Iran. EMousavi@Shahed.ac.ir, Tayebebeh.Khoshrooy@Shahed.ac.ir*
[2] *Azerbaijan State Oil and Industry University, Baku, Azerbaijan, Azar.Feyziyev@asoiu.edu.az*

*Correspondence: Ehsan Mousavi Khaneghah, Shahed University, Tehran, Iran. EMousavi@Shahed.ac.ir

Abstract

There is a possibility of dynamic and interactive nature occurring at any moment of the scientific program implementation process in the computing system. While affecting the computational processes in the system, dynamic and interactive occurrence also affects the function of the elements that make up the management element of the computing system. The effect of dynamic and interactive events on the function of the elements that make up the management element of the computing system causes the time required to run the user program to increase or the function of these elements to change. These changes either increase the execution time of the scientific program or make the system incapable of executing the program. The occurrence of dynamic and interactive nature creates new situations in the computing system that the mechanisms to deal with when designing the computing system are not defined and considered. In this paper, the Lazy-Copy process migration management mechanism, specifically the Lazy-Copy mechanism in distributed large-scale systems, the effects of dynamic and interactive occurrence in the computational system investigate, and the effects of dynamic and interactive occurrence on the system investigate. Computational processes on the migration process and vector algebras try to analyze and enable the Lazy-Copy process migration mechanism in support of distributed large-scale systems despite dynamic and interactive events

Keywords: Industry 4.0, Industry 5.0, Industry 6.0, Digital technology, Cosmetics industry, ICT

1. Introduction

In computing systems, the implementation of activities related to the process migration, because the migratory process at the time of suspension cannot respond to requests from other processes, increases the execution time of the scientific program (Vivek, V., et al., 2019; Mousavi Khaneghah, E., Noorabad Ghahroodi, R., & Reyhani ShowkatAbad, A., 2018; Yousafzai, A., et al., 2019). This issue makes several

mechanisms for the process migration based on the requirements of the scientific program in the field of process transfer patterns from the source computing element to the destination computing element, as well as reducing the suspension time and availability of the migratory process (Pickartz, S., Breitbart, J., & Lankes, S., 2016; Setiawan, I., & Murdyantoro, E., 2016, October; Morin, C., et al., 2003, August).

In traditional process migration systems, the process migration, during the implementation of activities related to the transfer of migratory process from the source to destination computing element, does not collect information about the status of the process, the status of the source and destination computing element and also factors affecting the process selection for process migration (Di, Z., Shao, E., & Tan, G., 2021; Stoyanov, R., & Kollingbaum, M. J., 2018, June). Traditionally in traditional computing systems, the process migration operates on a triple basis (origin, destination, process) and does not collect information about events in the computing system, especially events that affect process migration. The system status does not change after activating the load balancer and calling the process migration in traditional computing systems, and the system status does not change. The information sent to the process migration is valid in processing activities (Thoman, P., et al., 2018; Kumar, P., & Kumar, R., 2019; Xu, Y., et al., 2019). The crazy process migration mechanism develops to reduce the migration time of the migration process. There is no need to transfer the address space completely; code and program stack and transfer time Immigrant is independent of process size as an efficient mechanism (Tang, Z., et al., 2018; Shah, V., & Donga, J., 2020; Talaat, F. M., et al., 2020).

In the crazy process of migration, the lack of dependence on the size of the migratory process and the use of a single pattern to transfer different processes makes it possible to use the exact mechanism to transfer processes with ample address space. Independence of process size allows the load balancer to consider a broader range of candidate migration processes (Afzal, S., & Kavitha, G., 2019; Barak, A., & La'adan, O., 1998; Noshy, M., Ibrahim, A., & Ali, H. A., 2018). In most process migration mechanisms, because the size of the migratory process directly affects the suspension time and migration time, usually, the load balancer tries to select smaller processes for transfer and migration. The functional nature of process migration is insane in such a way due to the dependencies between the migrant process and other member processes of the source computational element until the execution of the process in the destination. Data connections maintain between the migratory process and other processes in the migration progress (Masdari, M., & Khoshnevis, A., 2020; Anawar, M. R., et al., 2018).

In distributed Exascale systems, dynamic and interactive events can occur in the scientific program implementation process (Anzt, H., et al., 2020; Ashraf, M. U., et al., 2018). The occurrence of dynamic and interactive nature and its effects on the system, especially on the functioning of the migration process, may either cause the migration cause to violate, cause the status of the source and destination computational element to change, or cause the migration time Increase the process. This event causes dynamic and interactive occurrences to occur in distributed Exascale systems. Their effects on process migration factors may cause the process of implementing migration activities to fail, or the source computational element allocates more than the sufficient time to perform migration-related activities to execute the migration. Changing the pivotal element of the process to the concept of global activity in distributed Exascale systems and the need for a mechanism that can handle Exa-sized processes with minimal downtime makes the lazy migration mechanism one of the most efficient mechanisms for the process migration (He, T., & Buyya, R., 2021).

The most critical challenge of applying the lazy migration mechanism in distributed Exascale systems is the high dependence on the status of the two computational elements of origin and destination (Stoyanov, R., & Kollingbaum, M. J., 2018, June; Khaneghah, E. M., et al., 2018; Chou, C. C., et al., 2019, November). One of the most important manifestations of the dependence of the lazy migration mechanism on the status of these two elements is in the management of memory pages. Suppose the status of the source computational element changes dynamically and interactively as a result of an occurrence in such a way that there is a need to call multiple pages. In that case, the lazy migration mechanism is ineffective. The lazy migration mechanism is such that it does not collect the status information of the two elements of origin and destination in the process of process migration, so it can not consider the effects of dynamic and interactive occurrences on the migration system (Ranjan, A., et al., 2015, March; Khaneghah, E. M., et al., 2011, December).

In this paper, while analyzing the function of the lazy process migration, the effects of dynamic and interactive occurrence on this function are investigated. This analysis makes it possible to develop a lazy process migration function based on the concept of determinism capable of managing dynamic and interactive impact effects.

2. Lazy Copy Functionality

In traditional computing systems, the lazy process migration, after being called by the load balancer, changes the status of the process from running to being suspended (Khaneghah, E. M., ShowkatAbad, A. R., & Ghahroodi, R. N., 2018, February; Plank, J. S., & Thomason, M. G., 2001; Yang, K., Gu, J., Zhao, T., & Sun, G., 2011, August). This change in status makes the process unable to respond to requests received and interact with other processes, like the process migration mechanisms defined in traditional computing systems. The process migration element transfers part of the process address space, file descriptors, and dirty memory pages immediately after changing the status of the process. The transfer of the above information set causes the initial processing space to form in the destination computational element. Based on the structure of the operating system used in the computing system, it can state that the initial state vector shown in Formula 1 is passed as the primary part of the process by the lazy process migration.

$$Process_{candid} = \langle < P_{u_1}, \dots, P_{u_m} >, < P_{r_1}, \dots, P_{r_n} > \rangle \quad (1)$$

As can be seen in Formula 1, the initial process description vector by the lazy process migration is a set of m available vectors, and n requested vectors. When transferring the status of the process from execution to suspension, the system management element allocates a set of resources to the process. Each resource used can be considered in the form of a vector in the size of the vector, the amount of resource used by the migratory process, and its direction have been in a positive direction.

If the classification used by the computing system management element follows the operating system classification for resources, then for each process, four P_{u_i} vectors can be defined, i can be the file, input, output, process, and memory.

By the load balancer, the migratory process may select for migration for two reasons: either the process requirements are not satisfied in the source computational element, or the destination computational element can provide the required resources in the shortest waiting time .In both cases, the purpose of the transfer of the process by the migration management element is to transfer the process to a computational

element that can meet the set of expected requirements of the process or< $P_{r_1}, \dots, P_{r_m}$ >. If the computing system management element uses the operating system classification pattern for resources, then four-vectors P_{r_i} can be defined as a file, input, output, process, and memory. The size of these vectors is the time required by the process to each source and its direction in a positive direction. From the point of view of the load balancer, each computational element is also equivalent to P_{r_i} Vectors in the form of four vectors. Such a definition allows the element of the load balancer to consider four different destinations in process migration for each process. However, the load balancer migrates the process to an element that meets all the process requirements in traditional computing systems.

The load balancer typically migrates the process to an element that meets all process requirements in traditional computing systems. The indicator should also describe the functional status and capability of the destination computational element to the requirements of the immigrant candidate process. Using the model described in Formula 1 and considering the process requirements as describing the status of the source and destination computational element provides the capability for a lazy process migration mechanism that describes the status of both elements based on the candidate process requirements. The dependence of the lazy process migration mechanism on the status of the source and destination computational element causes that in traditional computing systems, when the lazy process migration mechanism is activated, then the destination computational element must also satisfy the P_{u_i} vectors and also meet the requirements of the vectors P_{r_i}. Completing this condition is considered necessary for implementing lazy migration process activities.

In traditional computing systems, the load balancer must be able to make decisions based on an indicator (or indicators) about the ability of the destination computing element to respond to the candidate's process migration requests. In the lazy process migration mechanism, these indicators should define. In the lazy process migration mechanism, consider the status of the source and destination computational element as to which processes can qualify as a candidate migration process. As stated in Formula 1, the process described in the lazy process migration mechanism is described based on two vectors P_{u_i}andP_{u_i}, so the process migration mechanism must be able to be based on a general description pattern for each Generate two vectors P_{u_i}andP_{u_i}. This general description should include the adaptability of the destination computing element to the process requirements and the current requirements of the process met in the source computing element.

In the lazy process migration mechanism, based on the only requirement of the execution process is in the shortest possible time and the use of computational resources, and since the index is a function of the load balancer, the execution of the process in the shortest possible time and maximum use system. So in this type of computing system, In the lazy process migration mechanism, the process requirement is limited to the type of CPU source. In traditional computing systems, the description of the source and destination computing element by the load balancer and consequently the migration management element is a process based on the time allocated to the process in the source computational element and the time required by the process in the destination computing element. In Formula 1, the dimensions of the vectors $P_{u_i}$and P_{r_i} are one-dimensional and unique to the CPU, and formula 1 is rewritable as Formula 2.

$$Process_{candid} = \langle CPU_{Times}, Time_{conditions_s} >, < CPU_{time_d}, \dots, time_{condition_d} > \rangle \quad (2)$$

As seen in Formula 2, in traditional computing systems, the two vectors of current process requirements and process requirements that should meet in the destination computational element are described based on the two concepts of CPU usage time and process constraints. The concept of CPU usage time in the source computational element represents the time allotted to the process. Given that discrete times allocate to the processor in the computational element and the CPU allocation time is a scalar value, the load balancer uses to decide the outcome of the CPU time allocated to the processor and uses the weighted average. The processing and time constraints govern any process request access to the CPU.

The progress migration process should consider the Limitations required to allocate the CPU to the process, restrictions, and time limits to allocate the process to other processes. The constraints of assigning the CPU to the processor are required to execute the process to respond to the interactions of the process with other processes, including time constraints governing the process. Suppose the allocation of the process to the computational element is based on the initial allocation and the execution of the process starts from the local computational element. In that case, the system designer allocates the process requirements to the requirements of the process using the computing resource features in the local computational element.

In this case, it expects that the local computing element would be able to execute all the process requirements. In implementing the scientific program, due to changes in the requirements of other processes and especially the acceptance of global processes to run in the local computing element, some of these features can not be met in the local computing element. They may cause the Load balancer decision to migrate the process based on the lazy mechanism. In traditional computing systems, the conventional failure of the CPU to allocate to the process can usually result from the CPU being involved in running other processes and failing to meet the CPU's time constraints. According to the information collected by the load balancer, the migratory process transfers to the computational element that meets Formula 3.

$$\begin{bmatrix} Free\ time_{CPU_d} = \left(CPU\ time_{Time_{condition_s}} - CPU_{Time_s}\right) > 0\ and \\ \forall\ Request \in Remain\ \langle Time_{condition_s} \rangle \exists\ Answer \in time_{condition_d} \end{bmatrix} \quad (3)$$

As can be seen in Formula 3, in the lazy mechanism, the load balancer base two conditions: the ability to allocate the time required by the process and The ability to meet remaining constraints and time constraints as conditions for transferring the process from the computational element to the destination computational element. At the time of transferring the process from the source computing element to the destination computing element, two conditions must meet. Still, due to changes in the execution process of processes in the destination computing element and especially accepting the implementation of global processes, the double conditions mentioned in Formula number 3 are violated. Violation of any of the two conditions listed in Formula 3 requires the process to be re-transferred.

One of the essential advantages of the lazy process transfer mechanism is the absence of a process size constraint in Formula 3 for decision-making by the migration management element of the load balancer. In Formula 3, unlike other transfer mechanisms such as Total Copy, Pre Copy, and Flushing, the size of the migratory process has no effect on Formula 3 and is not used as a decision criterion for process transfer.

As can be seen in Formula 3, the status of each of the two computational elements of origin and destination plays a significant role in lazy process migration. The status of the set of constraints and constraints met as well as the amount of time allocated to the process as the primary and pivotal role of the source computing element and the ability to respond to the constraints of the process as well as provide the time to execute the process in the destination computational element. The title of the pivotal factor is the computational element of the destination. The lazy migration management element establishes the second condition of several mechanisms for estimating the constraints governing the CPU's use. Lazy migration uses to receive information from the user using PBS or program iterations and estimate based on program code iterations in the item using the time required to use the CPU. As can be seen in Formula 3, the executor of Formula 3 is the load balancer. The reason for this is that in traditional computing systems, the process migration element does not collect information about the status of the system and the status of the migratory process, the source, and the destination computing element. Formula 3 is written based on the characteristics of the lazy migration mechanism, but in Formula 3, the focus of the process is considered an abstract concept. In Formula 3, the process is considered a stand-alone concept with requirements, constraints, and constraints on accessing the CPU.

This event is due to the lack of change in the system's status, especially the computational element of origin, destination, and factors affecting the candidate migration process. The lazy process migration depends on the status of the computational element of origin and destination—formula 3 independent time and event variables. Independence of time and occurrence allows the load balancer to decide whether or not to make a lazy process migration based on a stable and unchanged situation.

3. Related Works

The lazy migration algorithm was first used in the Accent operating system to transfer the minimum address space required. The accent is a distributed operating system developed at CMU, and its program migration and process migration are such that it first used this technique to copy pages lazily. This strategy is an example of a paging request approach that should support remote paging. In this mechanism, only the process execution status transfer and the address space transfer as needed.

In accent, instead of copying the pages, the virtual parts are created on the destination machine, and when a pagination error occurs, these virtual parts create a link to the page on the source machine. Although not dependent on the size of the address space, this mechanism depends on the number of memory lags. Virtual memory transfer is the dominant cost in migration, so this method uses to reduce this cost by moving pages if needed. The articles (LaViola, J. J., Hachet, M., & Billinghurst, M., 2011, March; Al-Dhuraibi, Y., 2018) mention operating systems that use multi-strategy to transfer the process; the essential use of the lazy mechanism is the Mach operating system. This operating system, based on microkernels, has two tools for transferring processes. One is SMS, which uses a lazy mechanism for process migration, and the other is OMS, which provides user-level migration and uses lazy, pre-copy, and total transfer mechanisms.

Also mentioned in (Rough, J., & Gościński, A., 1998) is another type of operating system that uses a combination of several strategies to provide a single method to increase performance and reduce disadvantages. Algorithms derived from this

approach can refer to the generic algorithm, which uses three flushing algorithms: pre-copy and post-copy. That self-copying algorithm is an extension of the lazy algorithm. The generic algorithm includes three cycles before migration, during, and after migration. The delay of this algorithm is like a lazy mechanism. In the migration cycle, only the status of the process is transferred to the destination machine to continue running.

The backup mechanism has similarities to other algorithms. This mechanism is a combination of two mechanisms, pre-copywriter and lazy. Like the lazy algorithm, the execution of the process on the destination machine continues immediately after the migration decision. Unlike any algorithm, like a lazy mechanism, it only needs to transfer at least a subset of the process state to the destination machine. The process then sends a request to transfer the missed pages whenever it needs a page in the source machine, taking precedence over the regular transfer of pages. Therefore, the delay in using the post-copying algorithm, such as the lazy mechanism, is short.

Rough, J., & Gościński, A. (1998) refers to the RHODOS distributed operating system, which has two purposes. The first is to support distributed systems for parallel execution and load balancing, and the second is to compare different solutions to problems with distributed operating systems. To this end, RHODOS, developed as a flexible laboratory for distributed systems, has made process migration part of the operating system design phase. RHODOS Migration Center is designed to accommodate a variety of strategies. This event is confirmed by having process migration management. Real transfer mechanisms are also part of the appropriate kernel server. So lazy and direct copy mechanisms are part of the RHODOS management space. This configuration allows for flexible execution and control of many of the required strategies. Although this mechanism is not designed explicitly for RHODOS, it has been quickly added to the operating system with the ability to manage flexible memory in RHODOS.

4. Influence Dynamic and Interactive Events on Lazy Copy Functionality

The occurrence of dynamic and interactive nature in distributed macro-scale systems can change the status of each of the two computational elements of origin and destination and, consequently, the migratory process.

As stated in Formula 1, the immigrant candidate process can base on two vector spaces, P_{u_i} or the collection of process requirements from the source computational element, and P_{u_i} , the set of standard requirements of the candidate migration process from the computational element considered the destination. n traditional computing systems, from the point of view of the load balancer, the inadequacy of each of the P_{u_i}collection members or the inability to provide the P_{u_i}Collection members cause the process migration process to begin.

Also, in traditional computing systems, from the point of view of the load balancer, if the destination computing element can provide all members of the P_{u_i}set to the process candidate migration process, taking into account time and space constraints, then the acceptability capability and continues to run the process.

In traditional computing systems, the lazy process migration does not collect information about the status of either P_{u_i} and P_{u_i}. In traditional computing systems, any change in the state of the vector sets $P_{u_i}$and P_{u_i}After starting the lazy process; migration is not considered because the load balancer is not active. In traditional computing systems, at the moment of starting the process migration activity, the decision on the

implementation of the process migration activity, as well as the decision on the destination computing element and the status of the descriptive parameters of the source and destination computing element, is the load balancer and the set of decisions made by the load balancer. This event leaves the lazy migration manager lacking information about the decision-making process on migration. In computing systems, the implementation of process migration-related activities is the responsibility of the lazy process migration, so the load balancer is not activated and can not decide based on changes in the status of parameters governing the process migration process. In traditional computing systems, the lazy process migration, based on a set of specific parameters defined by the load balancer, executes the process of implementing activities related to process migration management.

In distributed Exascale systems, dynamic and interactive events can occur, including the start-up and decision-making process on migration. This event causes that in distributed Exascale systems, both load balancer and lazy process migration management need to describe the process status in the source and destination computational element.

In the lazy process migration mechanism, unlike other mechanisms used to manage process migration, the status of the source and destination computing element is described based on the requirements of the process. Following the requirements of the process, the minimum amount of data transfer to start the execution of the processing activity in the destination computational element.

The possibility of dynamic and interactive occurrence in distributed Exascale systems allows vector sets describing the process requirements of the source and destination computational element to be developed and rewritten according to Formula 4.

$$Process_{Candide_{D\&I}}^{Time}(G) = \langle [< P_{u_1}, \dots, P_{u_m} >](T, L, D, G), [< P_{r_1}, \dots, P_{r_n} >](T, L, Cap, G)\rangle \quad (4)$$

As can be seen in Formula 4, the candidate process changes state from an abstract process to a process described in the global activity. This state is because, in traditional computing systems, the load balancer considers the concept of process as an abstract concept that the failure of one or more members of the collection set P_{u_i}Causes that The process requires the transfer and migration of a process. The load balancer also considers the process concept in the destination computational element as an abstract concept. It assumes that if the collection members P_{u_i} are satisfied, then the process can be executed in It has the computational element of the destination. The load balancer also considers the concept of processing in the destination computational element as an abstract concept and assumes that if the collection members $P_{u_i}$are satisfied. In this case, the process can execute in the destination computational element.

In distributed Exascale systems, the concept of the process defines as a concept related to global activity. Each computational process defined in the distributed Exascalesystem is part of global activity in that member processes are activities in communication and interaction. The concept of process in distributed Exascalesystems influences the functionality and events governing other member global activity processes. Therefore, in distributed Exascalesystems, the concept of the migratory process cannot be considered without considering other member processes of global activity.

In Formula 4, This matter has caused the candidate migration process to be considered a function of other processes. From the point of view of the lazy process migration, the description of the migration candidate process and the process

requirements in the source computational element, and the process expectations of the destination computational element depend on the needs of other member activity processes.

In Formula 4, the candidate migration process concept is considered a process based on two independent variables of time and dynamic and interactive event occurrence. In distributed Exascalesystems, at any moment in implementing the candidate migration process, the process can change the status of the process descriptor parameters from the point of view of the system manager and the lazy process migration element. This event allows the candidate migration process to describes as a process based on an independent time variable. On the other hand, in distributed Exascale systems, at any point in implementing a candidate migration process, a dynamic and interactive event can occur and affect the descriptive elements of the process. This event has also led to considering a dynamic and interactive event occurrence independent variable to describe the status of a candidate migration process.

In Formula 4, the concept of collecting vector descriptors of process requests from the source computational element may also influence by the occurrence of a dynamic and interactive event. From the point of view of the system manager, the lazy process migration and the description of the processing requests from the computational source element may change due to a dynamic and interactive event occurring or the execution of the process migration activity. In other states, the lazy migration process leads to new activity. By changing the independent variables, the collection of process requirements from the computational source element may necessitate a review of the process of process migration activities. Collection of descriptive requirements of process requests based on independent variables constraint on response to process requests for each of the requested sources, location constraint on response to process requests, and the dependence of the process on the computational element of origin. It also defines the dependencies and interactions of the process with other member processes of the global activity. The occurrence of a dynamic and interactive event or a change in the execution time of a candidate migration process may change each of the four independent variable spaces describing the process requirements of the source computational element. The occurrence of a dynamic and interactive event may cause the temporal and spatial constraints governing the processing requests to change in such a way. It is either impossible to change the location of the process and the process of performing migration activities cancel, or need to implement new process migration activities based on new time and space constraints.

The occurrence of a dynamic and interactive event may cause the set of dependencies required by the process to respond to the source computational element to change. This dependency change may define new dependencies to meet the requirements of the process that did not exist at the time of the decision to start migrating the process or cause a set of dependencies to meet the requirements of the process. The process can consider a process migration candidate process.

A dynamic and interactive event may change the interaction and communication between the process's migratory candidate process and other processes instead of the global actuality of which the migration process is part. This change in interactions and connections between the CPU candidate process and other national activity member processes can violate the ability to select the process as a process skill candidate process or cause the process to have the ability to select as a candidate migration process.

The occurrence of a dynamic and interactive event may cause members of the

collection of processing requirements to change from the source computational element. The occurrence of a dynamic and interactive event may lead to the creation of vectors with a new type of demand source or violate vectors with an existing source type of requirement. For this reason, members of the processing requirements collection of the source computational element are also a function of two independent time variables and dynamic and interactive events.

In Formula 4, collecting victory descriptors of process requests from the destination computing element may also be influenced by the occurrence of a dynamic and interactive event. From the point of view of the system manager and the lazy process migration, the description of the processing requests from the destination computing element may change due to a dynamic and interactive event occurring. On the other hand, the process migration activity or the destination computing element cannot execute the process and meet its requirements. Maybe by changing the independent variables, the collection of process requirements from the destination computing element necessitates a review of the process of process migration activities.

Collection of descriptive requirements of requests based on independent variables. Time request for a response to request requests for each of the requested resources, location of response to request requests, and the capability of the destination computing element in responding. At the processor's request and consequently, the global activity related to the process and the global activity of which the processor is a part described. From the point of view of lazy process migration, when a process transfers from the source computing element to the destination computing element, the path of global activity changes. Redirection of global activities, in addition to affecting the performance of the candidate migration process, also affects the implementation of processes related to the candidate migration process. In addition to affecting the processes that make up global activity, redirection of global activity also affects the processes of other global activities related to global activity. These issues cause the destination computational element's capability to be considered one of the constraints and constraints affecting the process of process migration activity. From the point of view of the lazy process migration and the constraints and requests of the source type, the capability of the destination computational element concerning the global activity should consider.

A dynamic and interactive event causes each of the elements affecting the vector sets to describe the status of the computational element of origin and destination to cancel the continuation of the process migration activity. The occurrence of a dynamic and interactive event causes the need to start the process migration activity or develop a definition to create a computational element description.

5. Exa-Lazy Copy

The lazy process migration operates based on describing the status of the source and destination computational element based on the requirements of the process, so it must be able to index (or indexes) when a dynamic and interactive event occurs. Describe the needs and the changes made to them based on the occurrence of a dynamic and interactive event. The distribution management element typically manages the description of the processing requirements in the source computing element. Changing the descriptor requirements of the process requirements in the source computing element usually means deciding whether to activate the system manager and the lazy process migration to process the process.

In the event of a dynamic and interactive event affecting the collection parameters that describe the request requirements of the computational element of the process

migration activities cancel, there is a need for process migration. It is not or needs to decide based on new parameters about the start of the migration activity. Changing the parameters that describe the status of the process requirement of the source computational element does not change the process of lazy process migration activity. Only if the parameters describing the status of the process requirement from the computational element change the performance of the lazy process migration, which occurred after the decision and the start of the activities related to the process migration, occur.

Dynamic and interactive change parameters affect the process requirements of the source computational element. In this case, the source computing element can decide on the continuation of the process migration activities based on the computational element of the selected destination. It must be able to cancel the process of migration activities or start the migration activity. Decide on new processes and cancel existing processes. From the point of view of the lazy process migration, the occurrence of a dynamic and interactive event after the start of the process migration activity in each of the two computational elements of origin and destination means deciding on the possibility of continuing the process migration process. The system manager cancels migration-related activities. To this end, the process migration in the destination computing element must be able to make decisions based on an indicator (or indicators) about meeting the process space's current requirements that are not met in the source computational element. The lazy process migration defines the vector of the best Alpha approximation, representing the result of the quadratic vectors of the resources allocated to the migratory process in the source computational element.

To create the vector of the best Alpha approximation Exa-Lazy process migration mechanism creates the Request vector space, containing the quadratic vectors of the resources allocated to the migrating process in the source computing element. In traditional computing systems, the Request vector space has fix pattern. No events occur in the Request space that changes the pattern of the migratory process requirements in the source computing element. The only conceivable event in the Request space is the inability of the source computational element to respond to a Request member or to fail to meet the requirements of a Request space member. In distributed Exascalesystems, the Request space is a vector space that is a function of time and dynamic and interactive events. To change the state of the Request vector space and its constituent vectors., To define the Alpha vector, the Exa-Lazy process migration assumes that based on the P_{u_i}vectors, the Request space can be considered a subspace of the internal multiplication space of the P_{u_i}vectors, and the ideal vector is the best Alpha approximation in the Request space. The best Alpha approximation vector indicates that although a pattern governs the P_{u_i}vectors, the process request in the source computational element is responsive. There is no need to transfer the process from the source computational element based on the insane mechanism.

The possibility of defining the best Alpha approximation vector based on the P_{u_i} Vectors is because the crazy process migration mechanism does not depend on the size of the process. The vector generated by the best Alpha approximation is independent of the process size and is a computable process for each candidate migration process. The Request space represents the product of the P_{u_i}vectors, and their constituent elements are the vectors that represent the request-response scenarios. Each member of the Request space represents a template for responding to the request.

Because the Request space is finite, each member is a pattern of using the resources in the system with the transferor non-transfer of the process to execute the process. Based on this makes, it is possible to consider the orthogonal base B = ={$f_l, f_g, IO_l, IO_g, m_l, m_g, p_l, p_g$} which represents each of the four sources locally and globally. The orthogonal base ={$f_l, f_g, IO_l, IO_g, m_l, m_g, p_l, p_g$} describes the vectors that act as vectors that generate process response scenarios. The orthogonal base ={$f_l, f_g, IO_l, IO_g, m_l, m_g, p_l, p_g$} uses both local resources to respond to the request and transfers the request from the local level to the global level to use the resources of other computational elements.

For a candidate migration process, the best Alpha approximation vector can be calculated based on Formula 5.

$$\text{Alpha} = \sum_k (Beta|\, B_k) B_k \quad (5)$$

As shown in Formula 25, the Alpha vector expresses the best approximation and description of the status of the process requests answered in the computational element. This approximation indicates that, regardless of the type of source requested by the processor, what is the vector pattern for responding to process requests. In Formula 5, the Beta vector is a vector of the Request space that represents the ability to respond to the process in the local computational element without the need for migration. The Beta vector is defined based on the internal multiplication space of the P_{u_i}vectors. It represents the scenarios for using local resources in the source computational element, based on which the candidate migration requests can be processed. System answered in the source computational element. The Beta vector is the ideal vector for responding to process requests. If the process can provide answers in the local computational element without process migration, it is time to execute the Exa-Lazy or Lazy process migration. There is no need to reduce the time to run a scientific and applied program. In Formula 5, Alpha should ideally be in Beta. From the Exa-Lazy process migration perspective, a dynamic and interactive event did not occur in the source computing element. From the Exa-Lazy process migration perspective, the origin computing element responded to all requests related to the migratory process. From the perspective of the Exa-Lazy process migration, the effects of a dynamic and interactive event occurring on the performance of the source computational element are such that it does not interfere with the process of responding to the processing request.

Suppose the best Alpha approximation vector is not equal to the Beta vector. In that case, the Alpha vector is the best approximation for the Beta vector by the Request vectors, in which case the Beta-Alpha vector is perpendicular to any vector in the Request vector space. The Beta-Alpha vector indicates that based on the subtraction vector of acceptable vectors to respond to locally requested response scenarios, the vector can respond to the request using scenarios using resources outside the source computational element Immigrant candidate process.

Based on Formula 5, we can also calculate the vectors of the best Alpha approximation and the response vector to the migrant candidate processing requests in the destination computational element. Based on what is stated in formula 5, to obtain the Alpha and Beta vectors in the destination computational element, the Answer vector space can be defined based on the product of the multipliers P_{u_i}. Like the Request vector space, this vector space contains possible scenarios for responding to the process request in the destination computing element. Each member of the Answer space represents a template for responding to the migrant candidate process requests in the destination computing element.

Answer vector space, like Request vector space, is a finite space, and each member has a pattern of using the resource in the system, considering the response to part of the request in the source computational element or without considering the response to part of the request Includes the source in the computational element. Is it possible to define the orthogonal base γ $=\{f_l, f_g, IO_l, IO_g, m_l, m_g, p_l, p_g\}$, which represents each of the four sources locally and globally. In the orthogonal base γ, if the response scenario to a part of the request is not used in the source computational element, then the values of the bases p_g ،m_g،f_g and IO_gAre zero, and the bases mentioned in the description of the response scenarios to the process request.

In the destination computing element, the Beta vector means that the process enters into the destination computing element in responding to all (or part) of the candidate's migration request. This event requires the reprocessing of the process, including the return of the process that has not occurred to the computational element of the origin or other computational element. The beta vector is the ideal state to respond to the request of a candidate migration process in the destination computational element. In the Exa-Lazy process migration, like Lazy, the definition of the computational element is based on the description of the resources requested by the process, so the destination computational element is considered the ideal destination computational element from the perspective of the Exa-Lazy process migration element. It can respond to the process requirements vector, whether the source computing element has met any process requirements or part of it has been met by the source computing element.

According to the definition of the best approximation vectors for the source and destination computational element, the function of the Exa-Lazy process migration can be rewritten based on Formula 6 to Formula 3.

$$f(ExaLazy): \sum_k (Beta|\, B_k) B_k \overset{(Page_{number}, Page_{size}, time)}{\rightrightarrows} \sum_{\acute{k}} (\acute{Beta}|\, \gamma)\gamma_k \qquad (6)$$

As shown in Formula 3, the Exa-Lazy process migration element must be able to combine the two vectors of the best approximation of the source computing element and the destination computational element. These two vectors consider page size, the number of pages transferred, and the inability to meet the requirements of the candidate migration process in the source computational element. The destination computational element Must have the best approximation of the candidate's process requirements space. Formula 6 is a rewrite of Formula 3 based on the concept of mapping. Considering the vector concept is the best approximation for each of the four-vector spaces describing the computational element of origin and destination.

As can be seen in Formula 6, the mapping condition of the two vector spaces is the best approximation to each other, derived from the exceptional capabilities and features of the traditional Lazy process migration mechanism. In the Lazy process migration mechanism, the page size and the number of page transfers should keep to a minimum. Unlike other process migration mechanisms, the Lazy Copy process reduces the page transfer rate by moving the most miniature altered pages from the source-to-destination computing element. On the other hand, in the Lazy process migration mechanism, one of the essential features is allocating time to the migrating candidate process. in traditional computing systems, the concept of allocated time emphasizes the concept of allocated CPU time. In distributed Exascale systems, the concept of time allocated to each type of source emphasizes the process requirements that are not answered by the source computing element and passed to the destination computing element for the response.

As can be seen in Formula 6, if the vector of the best Alpha approximation in the

source element is equal to the Beta vector, the result of Formula 6 mapping is the mapping of the response space to the requirements of the candidate process. It results from responding to processing requests in the destination computing element. Similarly, it can show that if the vector of the best approximation of Alpha in the destination computational element is equal to the Beta vector, the mapping result necessarily leads to the completion of the process execution process in the destination computational element.

The process concept is not considered abstract or part of global activity in distributed Exascale systems. Considering the process as part of a global activity allows the process to have a set of interactions and connections between processes with other member processes of the global activity. This event makes the concept of Candidate Process Migration Requirements in distributed Exascalesystems in need of development more than traditional systems. Part of the requirements of the candidate migration process in distributed Exascalesystems is due to the requirements of the process and part of the interactions and communications and the definition of the process in the context of global activity. With this in mind, the definition of a candidate immigration process from the perspective of the migration management element of Exa Lazy processes can extend from Formula 1 to Formula 7.

$$Process_{candid}(page, time) = \begin{bmatrix} \langle P_{u_{1_l}}, \dots, P_{u_{m_l}} \rangle & \langle P_{u_{1_l}}, \dots, P_{u_{m_l}} \rangle \\ \langle P_{u_{1_g}}, \dots, P_{u_{m_g}} \rangle & \langle P_{u_{1_g}}, \dots, P_{u_{m_g}} \rangle \end{bmatrix}_{Event}^{Time} \quad (7)$$

As can be seen in Formula 7, the definition of the candidate's process space requirement is based on a 2 * 2 matrix whose first line describes the computational element of origin and destination based on the requirements of the process, and the second line describes the element. Origin and destination calculations are based on the requirements of the process definition in the global activity. Each process matrix element describing the candidate migration process is in the form of a vector set. Each member of the collection defining each row indicates the extent to which the process needs a specific resource in the source or destination computational element based on the process need or the defined need arising from the definition of the process in the global activity. As with Formula 1, classification of sources can provide for Formula 7, and a formula similar to Formula 2 can provide. The candidate process descriptor matrix is defined based on two variables of time and event. In traditional computing systems, the status of the two computational elements of origin and destination is described based on the requirements of the candidate migration process. The length of the process migration process does not change.

The system manager, t = Zeta, migrates processes based on the information received about the process requirements that the source computing element cannot respond to requests. The system manager, based on the information received by the user in Process Requirements Through application tools such as PBS, finds a destination computing element that can meet the processing requirements of a process migration candidate. Then, by calling the Lazy Process migration, process migration activities begin. In traditional computing systems, neither of the two elements of distribution management and lazy process migration management collects the status of changes made to the process requirements and consequently describes the source and destination computational element. They do not represent information and assume that the information collected at t = Zeta is valid until the end of the migration process. This event is even though in Exascale systems distributed at any point in implementing process migration activities, the possibility of a dynamic and interactive event occurring in the candidate migration process element. There are requirements

for this element and, consequently, a description of the source and destination computational element.

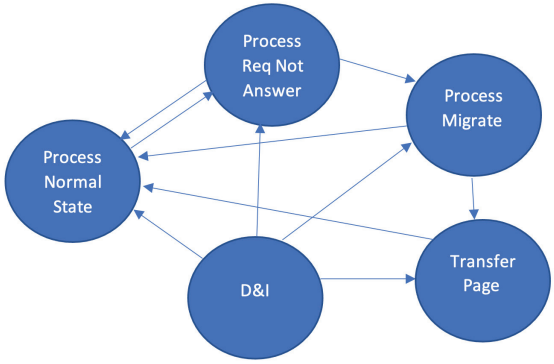

Fig. 1. Description of the functional status of the candidate migration process in the Exa-Lazy process migration

As shown in Figure 1, the process is usually in the normal state at the moment t = Zeta. Failure to respond to the process request in the computational element changes the process to the migration state and passes the executable program code. Based on the lazy process migration mechanism, attempts to transfer memory pages. In distributed Exascale systems, a dynamic and interactive event can also occur in addition to the process's normal state, leading to a change in the status of the process from the conventional process to the candidate migration process. Be considered a factor in changing the status of the process from the conventional process to the candidate migration process.

In distributed Exascalesystems, the occurrence of a dynamic and interactive event may also change the status of the process that is unable to respond to the process request in the computational element. This change may cause the current state of the inability of the source computational element to remain at the request of the process. The occurrence of a dynamic and interactive event eliminates the need for the computational element's process and changes the status to normal. In distributed Exascalesystems, dynamic and interactive event occurrences may also affect the status of the migrating process, which may cause the destination computing element to be unable to respond to the request. In this case, the description of the destination computational element has changed from the point of view of the process requirements, and the description of the status of the ability to respond to the request is not. In this situation, the process migration activity of the two situations of returning to the source computing element or calling the system manager and re-selecting the destination element based on the new situation can be imagined and considered.

On the other hand, the occurrence of a dynamic and interactive event may change the status of the immigrant candidate's process requirement. One can either return to the source computing element or call the distribution management element. Load and re-select the destination element. A dynamic and interactive event may also affect the page transfers during the migration process. The page transfer pattern between the source and destination computational elements does not follow the lazy process migration mechanism. In this case, the Exa-Lazy process migration element must be

able to change the mechanism according to changes in the page transfer pattern.

In Formula 7, the candidate migration process describes a process based on two independent variables, page and time. The time-independent variable involves executing activities that change the candidate migration process, and transfer refers to implementing process-related activities. Considering two independent variables of time and page allows the Exa-Lazy process migration to be aware of the status of the computational elements of origin, destination, and changes in the global activity of which the migratory process is a part.

References

Afzal, S., & Kavitha, G. (2019). Load balancing in cloud computing–A hierarchical taxonomical classification. *Journal of Cloud Computing, 8*(1), 1-24.

Al-Dhuraibi, Y. (2018). *Flexible framework for elasticity in cloud computing* (Doctoral dissertation, Université lille1).

Anawar, M. R., et al. (2018). Fog computing: An overview of big IoT data analytics. *Wireless Communications and Mobile Computing, 2018.*

Anzt, H., et al. (2020). Preparing sparse solvers for exascale computing. *Philosophical Transactions of the Royal Society A, 378*(2166), 20190053.

Ashraf, M. U., et al. (2018). Toward exascale computing systems: An energy efficient massive parallel computational model. *International Journal of Advanced Computer Science and Applications, 9*(2).

Barak, A., & La'adan, O. (1998). The MOSIX multicomputer operating system for high performance cluster computing. *Future Generation Computer Systems, 13*(4-5), 361-372.

Chou, C. C., et al. (2019, November). Optimizing post-copy live migration with system-level checkpoint using fabric-attached memory. In *2019 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)* (pp. 16-24). IEEE.

Di, Z., Shao, E., & Tan, G. (2021). High-performance Migration Tool for Live Container in a Workflow. *International Journal of Parallel Programming, 49*(5), 658-670.

He, T., & Buyya, R. (2021). A Taxonomy of Live Migration Management in Cloud Computing. *arXiv preprint arXiv:2112.02593.*

Khaneghah, E. M., et al. (2011, December). An efficient live process migration approach for high performance cluster computing systems. In *International Conference on Innovative Computing Technology* (pp. 362-373). Springer, Berlin, Heidelberg.

Khaneghah, E. M., et al. (2018). ExaMig matrix: Process migration based on matrix definition of selecting destination in distributed exascale environments. *Azerbaijan Journal of High Performance Computing, 1*(1), 20-41.

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).

Kumar, P., & Kumar, R. (2019). Issues and challenges of load balancing techniques in cloud computing: A survey. *ACM Computing Surveys (CSUR), 51*(6), 1-35.

LaViola, J. J., Hachet, M., & Billinghurst, M. (2011, March). Message from the symposium chairs. In *2011 IEEE Symposium on 3D User Interfaces (3DUI)* (pp. vii-vii). IEEE Computer Society.

Masdari, M., & Khoshnevis, A. (2020). A survey and classification of the workload forecasting methods in cloud computing. *Cluster Computing, 23*(4), 2399-2424.

Morin, C., et al. (2003, August). Kerrighed: a single system image cluster operating system for high performance computing. In *European Conference on Parallel Processing* (pp. 1291-1294). Springer, Berlin, Heidelberg.

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.

Noshy, M., Ibrahim, A., & Ali, H. A. (2018). Optimization of live virtual machine migration in cloud computing: A survey and future directions. *Journal of Network and Computer Applications, 110,* 1-10.

Pickartz, S., Breitbart, J., & Lankes, S. (2016). Implications of process-migration in virtualized environments. In *Proceedings of the 1st COSH Workshop on Co-Scheduling of HPC Applications* (p. 31).

Plank, J. S., & Thomason, M. G. (2001). Processor allocation and checkpoint interval selection in cluster computing systems. *Journal of Parallel and distributed Computing, 61*(11), 1570-1590.

Ranjan, A., et al. (2015, March). DyReCTape: A dynamically reconfigurable cache using domain wall memory tapes. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (pp. 181-186). IEEE.

Rough, J., & Gościński, A. (1998). *PVM on the RHODOS: A Preliminary Performance Study.* Deakin University, School of Computing and Mathematics.

Setiawan, I., & Murdyantoro, E. (2016, October). Commodity cluster using single system image based on Linux/Kerrighed for high-performance computing. In *2016 3rd International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)* (pp. 367-372). IEEE.

Shah, V., & Donga, J. (2020). *Load balancing by process migration in distributed operating system.* LAP LAMBERT Academic Publishing.

Stoyanov, R., & Kollingbaum, M. J. (2018, June). Efficient live migration of linux containers. In *International Conference on High Performance Computing* (pp. 184-193). Springer, Cham.

Stoyanov, R., & Kollingbaum, M. J. (2018, June). Efficient live migration of linux containers. In *International Conference on High Performance Computing* (pp. 184-193). Springer, Cham.

Talaat, F. M., et al. (2020). A load balancing and optimization strategy (LBOS) using reinforcement learning in fog computing environment. *Journal of Ambient Intelligence and Humanized Computing, 11*(11), 4951-4966.

Tang, Z., et al. (2018). Migration modeling and learning algorithms for containers in fog computing. *IEEE Transactions on Services Computing, 12*(5), 712-725.

Thoman, P., et al. (2018). A taxonomy of task-based parallel programming technologies for high-performance computing. *The Journal of Supercomputing, 74*(4), 1422-1434.

Vivek, V., et al. (2019). Payload fragmentation framework for high-performance computing in cloud environment. *The Journal of Supercomputing, 75*(5), 2789-2804.

Xu, Y., et al. (2019). Dynamic switch migration in distributed software-defined networks to achieve controller load balance. *IEEE Journal on Selected Areas in Communications, 37*(3), 515-529.

Yang, K., Gu, J., Zhao, T., & Sun, G. (2011, August). An optimized control strategy for load balancing based on live migration of virtual machine. In *2011 Sixth Annual ChinaGrid Conference* (pp. 141-146). IEEE.

Yousafzai, A., et al. (2019). Process migration-based computational offloading framework for IoT-supported mobile edge/cloud computing. *IEEE internet of things journal, 7*(5), 4171-4182.

Submitted: 17.08.2021
Accepted: 03.11.2021