US 20230244545A1

(19) **United States**

(12) **Patent Application Publication**

GUO et al.

(10) **Pub. No.: US 2023/0244545 A1**

(43) **Pub. Date: Aug. 3, 2023**

(54) **RELIABILITY-AWARE RESOURCE ALLOCATION METHOD AND APPARATUS IN DISAGGREGATED DATA CENTERS**

(71) Applicant: **City University of Hong Kong**, Hong Kong (HK)

(72) Inventors: **Chao GUO**, Hong Kong (HK); **Xinyu WANG**, Hong Kong (HK); **Moshe ZUKERMAN**, Hong Kong (HK)

(21) Appl. No.: **17/586,818**

(22) Filed: **Jan. 28, 2022**

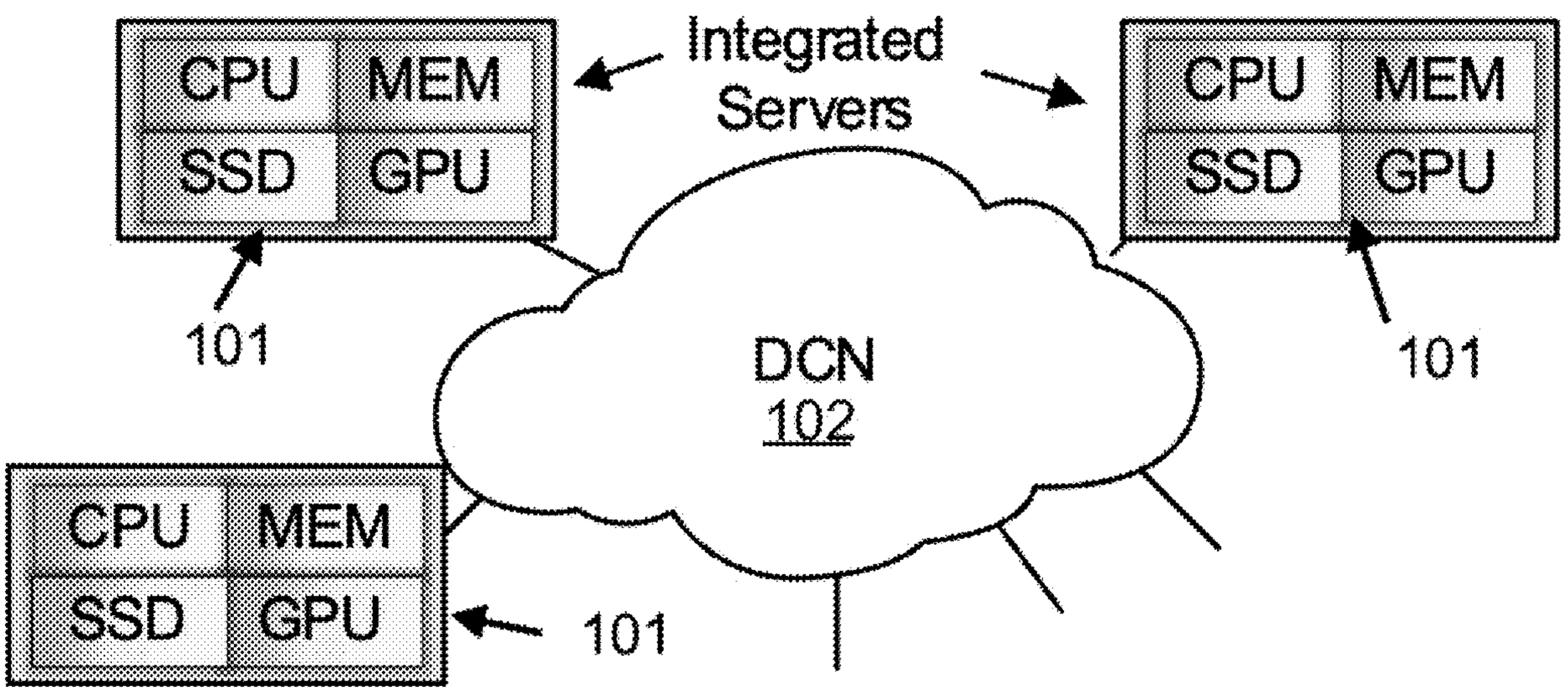
Publication Classification

(51) **Int. Cl.**
G06F 9/50 (2006.01)

(52) **U.S. Cl.**
CPC ***G06F 9/5083*** (2013.01)

(57) **ABSTRACT**

A method for resource allocation in a disaggregated data center (DDC), comprising: a reliability model to determine an achievable reliability for a service request to the DDC; a integer linear programming (ILP) model to perform a resource allocation for the service request to the DDC such that maximizing total number of service requests received by the DDC accepted for execution is maximized, while the number of the accepted service requests allocated with backup computing resources is minimized; and a heuristic process to perform a resource allocation for the service request to the DDC such that the least reliable node of each needed computing resource type is allocated but still meeting the reliability requirement of the service request.



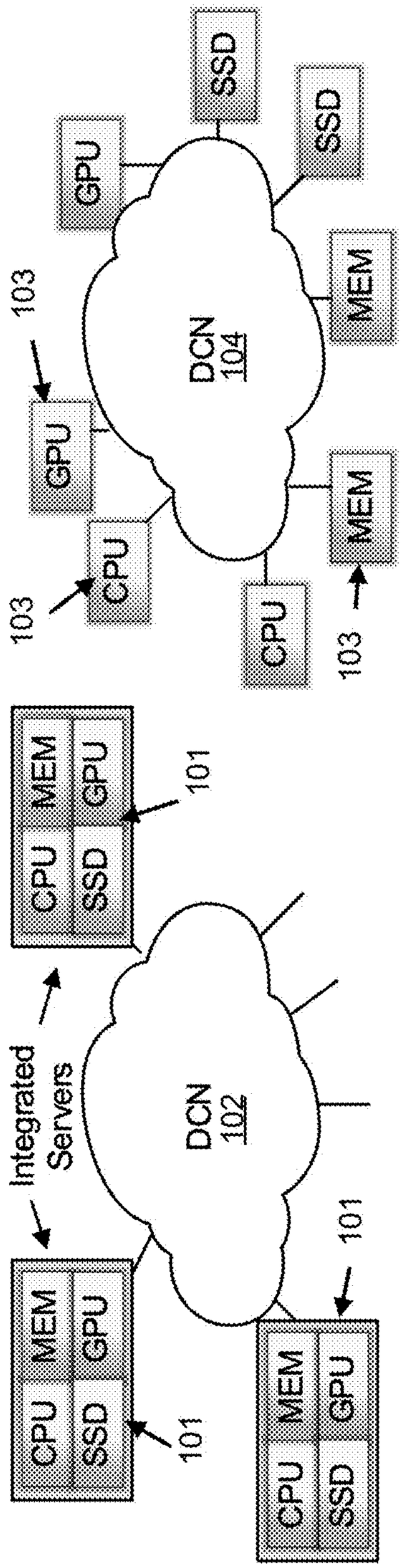

Integrated Servers
101
CPU
MEM
GPU
SSD
DCN
102
101
CPU
MEM
GPU
SSD
101
CPU
MEM
GPU
SSD
DCN
104
103
CPU
GPU
103
CPU
103
MEM
MEM
SSD
SSD
GPU

FIG. 1A

FIG. 1B

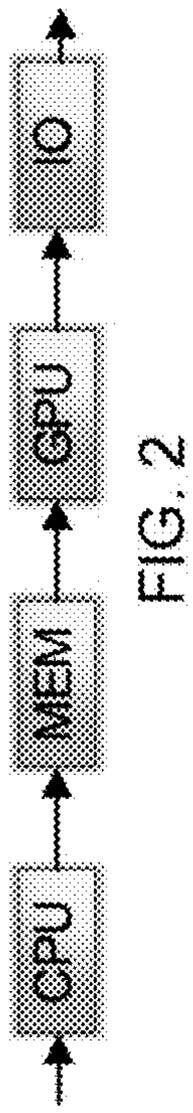


FIG. 2

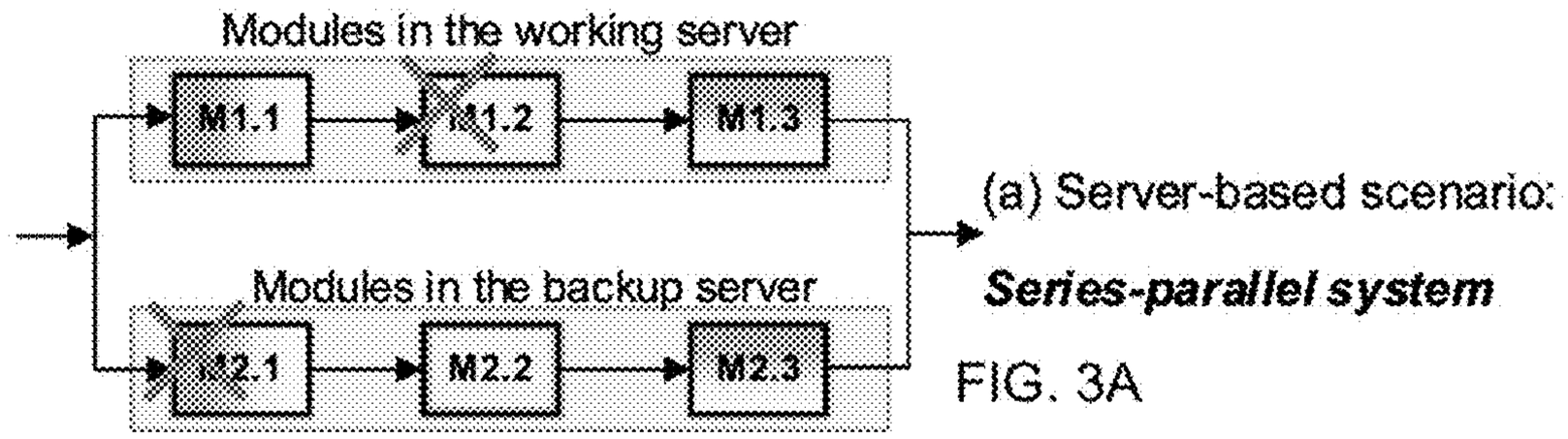


FIG. 3A

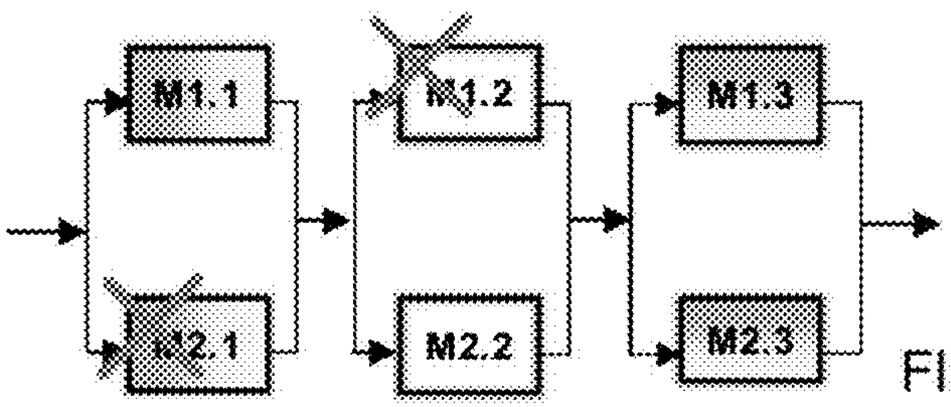


FIG. 3B

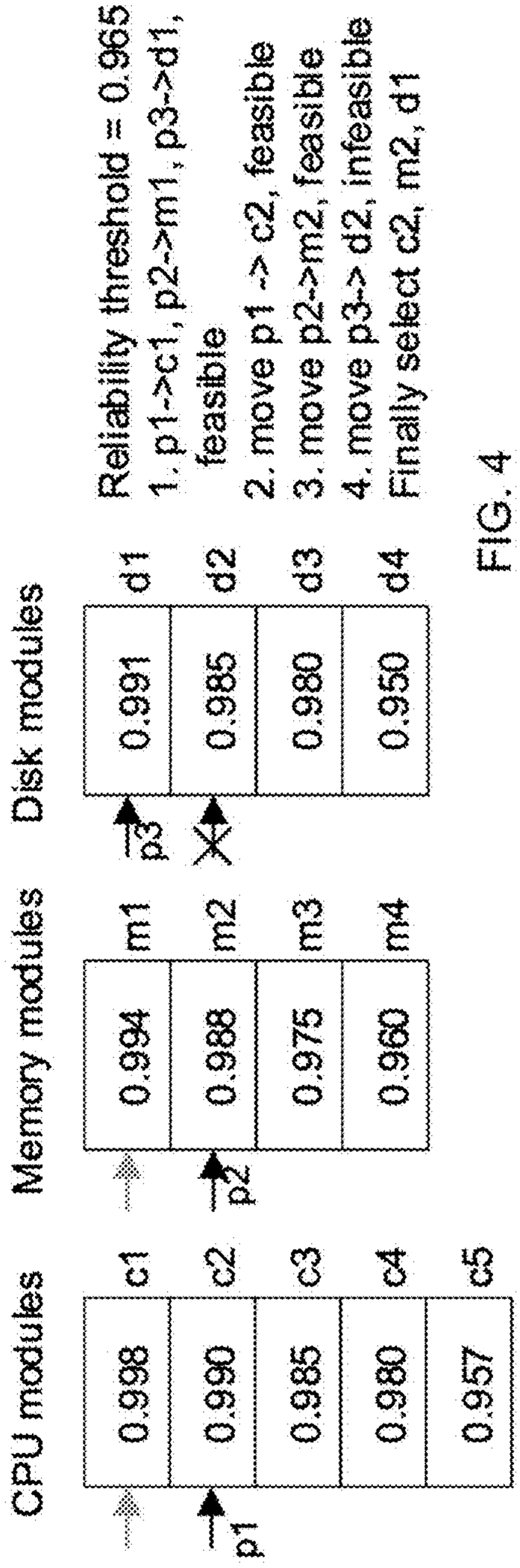

CPU modules
c1
c2
c3
c4
c5
0.998
0.990
0.985
0.980
0.957
p1
Memory modules
Disk modules
m1
m2
m3
m4
d1
d2
d3
d4
0.994
0.988
0.975
0.960
0.991
0.985
0.980
0.950
p2
p3
Reliability threshold = 0.965
1. p1->c1, p2->m1, p3->d1, feasible
2. move p1 -> c2, feasible
3. move p2->m2, feasible
4. move p3-> d2, infeasible
Finally select c2, m2, d1

FIG. 4

RELIABILITY-AWARE RESOURCE ALLOCATION METHOD AND APPARATUS IN DISAGGREGATED DATA CENTERS

COPYRIGHT NOTICE

FIELD OF THE INVENTION

[0002] The present invention generally relates to the resource management in data centers. More specifically, the invention relates to techniques of allocating resources to improve reliability of disaggregated data centers.

BACKGROUND OF THE INVENTION

[0003] Data centers (DCs) are typically built using one or more clusters of computer servers, each of which tightly integrates various resources such as CPUs, GPUs, transient and non-transient memory circuitries as illustrated in FIG. **1**A. These integrated computer servers **101** are further interconnected by a dedicated network inside the DC using a DC network (DCN) **102**, that is typically implemented using very high throughput data network components such as highspeed data network switches and optical fibers. Due to the close coupling of different types of resources in each computer server, when one type of resource in a server is exhausted, other resources also become unavailable. This tight resource coupling also makes it difficult to upgrade the integrated computer server. This is the reason why while the resource components in a computer server often have different lifecycles, hardware upgrade and expansions are typically done only at the computer server level. Such an architecture is not only resource-wasteful, it is inefficient for diversified workloads.

[0004] In addressing the aforementioned shortcomings and improve resource utilization, upgradability, and scalability, resource disaggregation is becoming a tread in the art. As illustrated in FIG. **1**B, resource disaggregation in a disaggregated data center (DDC) separates different types of resources in each computer server into distinct nodes **103** and then interconnects these nodes using a DCN **104** having high throughput and low latency. The management of these nodes, however, presents a new set of challenges.

SUMMARY OF THE INVENTION

[0005] It is an objective of the present invention to provide a method and an apparatus for resource disaggregation that not only provides better utilization, easier upgradability and scalability of computing resources, but also improves reliability of a disaggregated data center (DDC). Embodiments of the present invention improve the flexibility in resource allocation such that appropriate group of nodes (or modules of computing resources) can be selected to meet each service request's reliability requirements. Resource utilization of the DDC is improved because of the decoupling of computing resources provided by the resource disaggregation, which prevents chained failures (e.g., failure of one type of resource forcing the entire computer server to fail and making other types of resources in that computer server unusable). Furthermore, the resource disaggregation improves the reliability of the DDC by reducing the failure domain to allow backups and replicas to be allocated more efficiently.

[0006] In accordance to one aspect of the present invention, a reliability model for determining a degree of reliability of executing a service request to a DDC implemented with the resource allocation method or the apparatus in accordance an embodiment of the present invention is provided. The service request is provisioned either with the DDC's working resources alone or also with the DDC's backup resources. Based on the reliability model, the resource disaggregation method comprises a reliability-aware resource allocation scheme. In accordance to one embodiment, the service request is first attempted to be allocated to only working resources; if the reliability requirement of the service request cannot be met, the service request is then attempted to be allocated to backup resources; and finally, if the reliability requirement of the service request cannot be satisfied, the service request is rejected. The reliability-aware resource allocation scheme also attempts to allocate highly reliable resources to those service requests with high-reliability requirements.

[0007] In accordance to another aspect of the present invention, an integer linear programming (ILP) model for reliability-aware resource allocation in a DDC implemented with the resource allocation method or the apparatus in accordance an embodiment of the present invention is provided. The ILP model is to maximize the number of accepted service requests to the DDC, while minimizing the number of service requests provisioned with backup resources, aiming to guarantee reliability with only working resources.

[0008] In accordance to yet another aspect of the present invention, a heuristic process for reliability-aware resource allocation in a DDC implemented with the resource allocation method or the apparatus in accordance an embodiment of the present invention is provided. The heuristic process is scalable and has a lower complexity than the ILP model.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] Embodiments of the invention are described in more details hereinafter with reference to the drawings, in which:

[0010] FIG. **1**A depicts a schematic diagram illustrating an exemplary DC with integrated computer servers;

[0011] FIG. **1**B depicts a schematic diagram illustrating an exemplary DDC with nodes;

[0012] FIG. **2** depicts a schematic diagram illustrating a service request being executed by an exemplary series system;

[0013] FIG. **3**A depicts a schematic diagram illustrating an execution of a service request to a DC under a server-based scenario;

[0014] FIG. **3**B depicts a schematic diagram illustrating an execution of a service request to a DDC under a disaggregated scenario; and

[0015] FIG. **4** depicts an illustrative diagram of exemplary settings and conditions of an execution of service request to a DDC implemented with a heuristic process in accordance to one embodiment of the present invention.

DETAILED DESCRIPTION

[0016] In the following description, methods and apparatuses for resource allocation in DDCs, and the likes are set forth as preferred examples. It will be apparent to those skilled in the art that modifications, including additions and/or substitutions may be made without departing from the scope and spirit of the invention. Specific details may be omitted so as not to obscure the invention; however, the disclosure is written to enable one skilled in the art to practice the teachings herein without undue experimentation.

[0017] For simplicity and better illustration of the embodiments of the present invention, the following assumptions are made. In a DDC implemented with the resource allocation method or the apparatus in accordance an embodiment of the present invention, the DCC comprises one or more computer servers; each computer server contains only one hardware module of each type of resource, e.g., one CPU module and one memory circuitry; and each module has a specific resource capacity, e.g., one CPU may contain 16 cores. In practice, a computer server may have more than one module for a specific type of resource, e.g., two CPU modules; in the following description, however, all of them are regarded as a single module with a capacity equaling the total capacity of all these original modules. Further, a fully disaggregated scenario and that each resource module in a computer server becomes a distinct node after disaggregation are assumed.

[0018] In addition, service requests to a computer server in a DCC may include requests for virtual machines (VMs), virtual containers, and applications. Each service request is specified by its resource demand and reliability requirement. Although resource disaggregation allows a service request to use more resources than a traditional computer server can provide, for simplicity and better illustration of the embodiments of the present invention, it is assumed that each service request uses a certain resource from one module (e.g., as a traditional computer server can provide) in the disaggregated scenario.

[0019] In accordance to one aspect of the present invention, a reliability model for determining a degree of reliability of executing a service request to a DDC implemented with the resource allocation method or the apparatus in accordance an embodiment of the present invention is provided.

[0020] The reliability of a resource module is defined as the probability that the module will perform its function during a given period. Apart from this definition, other metrics that can quantitatively measure hardware reliability include the mean time to failure (MTTF), mean time between failure (MTBF), failure in time (FIT), and failure probability. For commercial hardware, the MTBF is often given in their specifications, e.g., the MTBF, FIT, etc. Given the MTBF, the reliability at any time can be estimated with the aid of a particular stochastic process assumption, e.g., exponential or Weibull distribution. There are also other ways to estimate the reliability values, e.g., Bayesian network. Many factors may affect the reliability, e.g., the hardware manufacturing level, age, and carried load. The reliability of a given resource module generally decreases with time as the fault probability often grows when aging. For simplicity and better illustration of the embodiments of the present invention, a static scenario is assumed, and it is further assumed that the service completion time of all given service requests to the DCC is negligible compared with the MTBF of a resource module. Under this situation, hardware reliability can be seen as a constant for all service requests. In any case, at any point of time, e.g., at a time when a service request is received, the DCC, as well as each of the computer servers, contain certain number of working resources and certain number of backup resources.

[0021] When a service request is scheduled on a traditional server, all the resources need to be available to guarantee a normal service. The service request is modeled as a series system as illustrated in FIG. **2**, and a failure of any of the resources results in the failure of the service request. This is also applicable to the disaggregated scenario, where any module allocated to the request should be operational to ensure its service. Thus, in both the server-based and disaggregated scenarios, the achieved reliability $\mathcal{R}$ of the request, e.g., the probability that the request functions normally, is the product of the reliabilities of its required modules, given by:

$$\mathcal{R}=\Pi_{r\in R}\mathcal{R}_{\mathcal{M}_r} \tag{1}$$

[0022] where R is the set of resource module types, which may include CPU, memory circuitry, etc.; $\mathcal{M}_r$ is the resource module of the resource type r allocated to the service request; and $\mathcal{R}_{\mathcal{M}_r}$ is the reliability of the module $\mathcal{M}_r$, e.g., the probability that resource module $\mathcal{M}_r$ functions normally.

[0023] If only allocating the working resources cannot meet the reliability requirement of a service request, the backup resource is allocated to improve the reliability. FIGS. **3**A and **3**B illustrates the different reliabilities of the service request for the server-based and disaggregated scenarios respectively.

[0024] In the server-based scenario, backup resources are provided at the level of a server, which can be modeled as a series-parallel system as illustrated in FIG. **3**A. First, in each of the working and backup servers, modules of different resource types form a series system, where all modules in a server should normally work to avoid server failure. Then, the two series systems, e.g., M1.1-M1.2-M1.3 and M2.1-M2.2-M2.3, form a parallel system because as long as one of them is operational, the request can be accommodated. Accordingly, the level of reliability is obtained by:

$$\mathcal{R}=1-(1-\Pi_{r\in R}\mathcal{R}_{\mathcal{M}_r^W})\cdot(1-\Pi_{r\in R}\mathcal{R}_{\mathcal{M}_r^B}); \tag{2}$$

[0025] where $\mathcal{M}_r^W$ and $\mathcal{M}_r^B$ are the modules of resource type r in the working and backup servers, respectively. The first product term, e.g., $(1-\Pi_{r\in R}\mathcal{R}_{\mathcal{M}_r^W})$, is the failure probability of the working server, where $\Pi_{r\in R}\mathcal{R}_{\mathcal{M}_r^W}$ is the reliability of the working server, according to (1). Similarly, the term $(1-\Pi_{r\in R}\mathcal{R}_{\mathcal{M}_r^B})$ is the failure probability of the backup server. The product of the two failure probabilities is the probability that both servers fail.

[0026] In the disaggregated scenario, the backup is at the level of a single module. As illustrated in FIG. **3**B, the service request can be modeled as a parallel-series system. Different modules of the same type form a parallel system, and for each resource type, as long as one module of this resource type works, the resource is available. Furthermore, all types of resource modules should be available to avoid service failure. Therefore, resource modules of different

types further form a series system. The achievable reliability is the product of the reliabilities of all types of resources, as:

$$\mathcal{R}=\Pi_{r\in R}(1-(1-\mathcal{R}_{\mathcal{M}_r^W})\cdot(1-\mathcal{R}_{\mathcal{M}_r^B})); \quad (3)$$

[0027] where $1-(1-\mathcal{R}_{\mathcal{M}_r^W})\cdot(1-\mathcal{R}_{\mathcal{M}_r^B})$ is the reliability of the pair of modules $\mathcal{M}_r^W$ and $\mathcal{M}_r^B$, which are of resource type r. Such a module pair follows a parallel system model, and only when the two modules fail simultaneously does the resource of type r become unavailable.

[0028] The reliability model for determining a degree of reliability of a service request to a DDC implemented with the resource allocation method or the apparatus, therefore, comprises the computation of equation (3) above to obtain the achievable reliability of a service request to a DDC.

[0029] The example illustrated in FIGS. **3**A and **3**B shows that resource disaggregation can achieve higher reliability due to the lower backup granularity. For example, assume that M2.1 and M1.2 in the disaggregated scenario fail simultaneously, then the service request's resource demand can still be satisfied by the remaining modules in the disaggregated scenario, but the service request fails in the server-based scenario. Assume that in the example illustrated in FIGS. **3**A and **3**B, each module has the same reliability of 99.5%. According to (2), the achievable reliability in the server-based scenario is 99.9777241%. According to (3), the achievable reliability in the disaggregated scenario is 99.9925002%, which is 0.0147761% higher than the former.

[0030] The reliability model (3), however, does not apply to legacy applications or VMs because they are not disaggregation-aware and do not support component-level failure independence. For example, in a VM, as long as even one single module used by this VM fails, all used resources will be isolated. Therefore, for legacy applications and VMs, the reliability model is the same as the server-based scenario. To maximize the reliability performance benefit brought from failure independence, significant evolutionary upgrades, e.g., RAID-style replication, in application protocol design and operating system (OS) models are needed, but they are considerably more complicated. For simplicity and better illustration of the embodiments of the present invention, it is assumed that service requests are disaggregation-aware and support failure independence.

[0031] In one embodiment of the DDC implemented with the resource allocation method or the apparatus in accordance an embodiment of the present invention, the DDC comprises a central controller responsible for global resource management. The DDC further comprises one or more lower-level (e.g., rack-level) systems for monitoring the state information of hardware modules, including module load, reliability and health condition, and failure occurrence, and report them to the central controller. The lower-level systems create one or more instances for a service request received based on the allocation results received from the central controller. For a service request provisioned with backup resources, each backup module (e.g., CPU) is a standby counterpart of a working module. The DDC, the computer servers, and/or modules are implemented with certain protocols to support fast recovery. For example, when implementing RAID-style replication for tolerating memory fault, the lower-level systems create a memory replica of the working memory module on the backup memory module. The lower-level systems also mirror the identical traffic from the nodes to the replica. For another example, the checkpointing technique may be employed by the lower-level systems for recovering from CPU failures. In this case, the lower-level systems checkpoint the CPU state, e.g., states of registers and program counter. The state information is then stored in the assigned memory for fast recovery. In addition, when a failure occurs in one of the working modules, the lower-level systems switch the running applications to the corresponding backup module, while other working modules keep serving the service request without interrupting the service.

[0032] In accordance to another aspect of the present invention, an ILP model for reliability-aware resource allocation in a DDC implemented with the resource allocation method or the apparatus is provided. The objective of the ILP model is to maximize the number of accepted service requests to the DDC, while minimizing the number of service requests provisioned with backup resources, aiming to guarantee reliability with only working resources.

[0033] In one embodiment, ILP model comprises computing the maximum of:

$$\Sigma_{i\in I}\omega_i-\epsilon\cdot\Sigma_{i\in I}\chi_i; \quad (4)$$

[0034] where $\Sigma_{i\in I}\omega_i$ is the number of accepted service requests; $\Sigma_{i\in I}\chi_i$ is the number of accepted service requests provisioned with backup resources; and E is a weight factor. When $\epsilon=1$, the objective of the ILP model turns into maximizing the number of accepted service requests that are not allocated with backup resources. To give a higher priority to maximization of the number of accepted service requests, the weight factor ϵ is to be set to a small number, e.g., $\epsilon=0.001$.

[0035] The objective of the ILP model is, however, subject to the following constraints:

$$\sigma_i = \Sigma_{m\in M_r}\delta_m^{ir}\forall\, i\in I, r\in R; \quad (5)$$

$$\chi_i = \Sigma_{m\in M_r}\gamma_m^{ir}\forall\, i\in I, r\in R; \quad (6)$$

$$\delta_m^{ir}+\gamma_m^{ir}\le 1\forall\, i\in I, r\in R, m\in M_r; \quad (7)$$

$$\sigma_i\ge\chi_i\forall\, i\in I; \quad (8)$$

$$\omega_i=\sigma_i\forall\, i\in I; \quad (9)$$

$$\Sigma_{i\in I}\left(\delta_m^{ir}+\gamma_m^{ir}\right)\cdot D_{ir}\le C_{rm}\forall\, r\in R, m\in M_r; \quad (10)$$

$$\Sigma_{r\in R}\Sigma_{m\in M_r}\xi_m^{ir}\cdot\log\mathcal{R}_{rm}+ \quad (11)$$

$$\Sigma_{r\in R}\Sigma_{m,n\in M_r}\mu_m^{ir}\cdot\log\left(1-\left(1-\mathcal{R}_{rm}\right)\cdot\left(1-\mathcal{R}_{rn}\right)\right)\ge\omega_i\cdot\log\,\theta_i\forall\, i\in I;$$

$$\begin{cases}\mu_{mn}^{ir}\ge\delta_m^{ir}+\gamma_n^{ir}-1\\ \mu_m^{ir}\le\delta_m^{ir}\\ \mu_{mn}^{ir}\le\gamma_n^{ir}\end{cases}\quad\forall\, i\in I, r\in R, m, n\in M_r; \quad (12)$$

$$\begin{cases}\xi_m^{ir}\ge\delta_m^{ir}-\chi_i\\ \xi_m^{ir}\le\delta_m^{ir}\\ \xi_m^{ir}\le 1-\chi_i\end{cases}\quad\forall\, i\in I, r\in R, m\in M_r; \quad (13)$$

[0036] where the variables are defined as follow:

	Sets:
R	Set of server resource types, e.g., CPU, GPU, and memory.
M_r	Set of resource modules associated with resource r.
I	Set of resource allocation requests.
	Parameters:
C_{rm}	The amount of available capacity in module m of resource r.
R_{rm}	Reliability of resource module m of resource type r.
D_{ir}	Resource demand of request i for resource r.
θ_i	Reliability requirement of request i.
ϵ	A real number weight factor.
	Decision variables:
δ_m^{ir}	A binary variable that equals one if module m of resource type r is allocated to request i for providing the working resource (this module is referred to as i's working module); zero, otherwise.
γ_m^{ir}	A binary variable that equals one if module m of resource type r is allocated to request i for providing the backup resource (this module is referred to as i's backup module); zero, otherwise.
σ_i	A binary variable that equals one if request i is successfully allocated with working resources; zero, otherwise.
χ_i	A binary variable that equals one if request i is successfully allocated with backup resources; zero, otherwise.
ω_i	A binary variable that equals one if request i is accepted; zero, otherwise.
μ_{mn}^{ir}	A binary variable that equals one if modules m and n of the same resource type r are allocated to request i as its working and backup modules, respectively; zero, otherwise.
ξ_m^{ir}	A binary variable that equals one if request i is allocated with only working resources without backup, and module m of resource type r is allocated to i as its working module.

[0037] Constraints (5) and (6) ensure that each service request can get a working (or backup) resource of type r from only one module. For constraint (5), when the left part $\sigma_i=1$, the right part must be one, meaning that for all possible modules, there is one and only one of them satisfying the requirement. Constraint (7) ensures that a service request's working and backup resources cannot be shared by a module. Constraint (8) ensures that if a service request is not allocated with working resources, backup resources will not be allocated to it. On the other hand, if the service request is allocated with backup resources, it must be allocated with working resources. Constraint (9) ensures that when a service request is successfully allocated with working resources, it is accepted, regardless of whether it is allocated with backup resources or not. Constraint (10) states the physical capacity restriction. For each resource module m, the total resource demand of all of the service requests held (allocated and pending for execution) for m cannot surpass the capacity of m. Constraint (11) ensures that the reliability requirements of each accepted service request should be fulfilled. The first term on the left side of the inequality $\Sigma_{r\in R}\Sigma_{m\in M_r}\xi_m^{ir}\cdot\log\ \mathcal{R}_{rm}$ corresponds to the situation that service request i is allocated with only working resources, while the second term corresponds to the situation that i is allocated also with backup. In constraint (11), models (1) and (3) are not directly applied because the product terms will be involved introducing nonlinear constraints. Instead, log function is introduced to the models (1) and (3), turning the product terms into summation terms. Given a reliability threshold θ_i, the inequalities using models (1) and (3) are equivalent to below inequalities (14) and (15), respectively. In constraint (11), the left side of the inequality contains both two situations considering whether the service request is allocated with or without backup resources, but the two situations should not happen simultaneously.

$$\Sigma_{r\in R}\log\ \mathcal{R}_{\mathcal{M}_r}\geq\log\theta_i \tag{14}$$

$$\Sigma_{r\in R}\log(1-(1-\mathcal{R}_{\mathcal{M}_r^W})\cdot(1-\mathcal{R}_{\mathcal{M}_r^B}))\geq\log\theta_i. \tag{15}$$

[0038] Constraint (12) is equivalent to $\mu_{mn}^{ir}=\delta_m^{ir}\cdot\gamma_n^{ir}$, but in a linear approach, ensuring that only when modules m and n of the same resource type r are simultaneously allocated to service request i as its working and backup modules, μ_{mn}^{ir} equals one. Constraint (13) is equivalent to $\xi_m^{ir}=\delta_m^{ir}\cdot(1-\chi_i)$, e.g., when service request i is not allocated with backup modules but selects module m of type r as one of its working modules, the binary indicator ξ_m^{ir} equals one.

[0039] The computational complexity of the ILP model in terms of both the dominant variable number and the dominant constraint number are $O(|I|\cdot|R|\cdot|M_r|^2)$, due to the variable μ_{mn}^{ir} and constraint (12), respectively.

[0040] The ILP model is unsolvable for large DDC due to its high complexity. In accordance to yet another aspect of the present invention, a heuristic process for reliability-aware resource allocation in a DDC implemented with the resource allocation method or the apparatus is provided. In this algorithm, the number of accepted service requests to the DDC is maximized from two aspects. Firstly, the reliability requirement of each service request is satisfied by allocating working resources, and only when the requirement is violated will backup resources be attempted to be allocated. Secondly, in order to guarantee the reliability of a certain service request, the reliability of modules assigned to this service request should be as high as possible. Nevertheless, this will affect the acceptance of other service requests, especially those with higher reliability requirements. Therefore, modules will be attempted to be allocated to each service request that is least reliable but reliable enough to satisfy its requirement.

[0041] In one embodiment, the heuristic process is represented by the pseudocode, Algorithm 1, listed below:

Algorithm 1

```
Input: i, a request; M_r∀r ∈ R, sets of resource modules.
Output: An integer value that equals 1 if i is accepted without backup
        allocation, 2 if accepted with backup allocation, and −1 if rejected.
1.  For ∀r ∈ R
2.    Exclude modules in M_r with insufficient resources;
3.    L_r ← Modules of type r in descending order of reliability;
4.  If (Trial (i,L_r,1)) // Allocate with no backup
5.    Return 1; //Success without backup
6.  If (Trial (i,L_r,2)) //Allocate with backup
7.    Return 2;//Success with backup
8.  Return −1; //Failure
Function: Trial (i,L_r,K)
Input: i, a request;
       L_r∀r ∈ R, lists of modules in descending order of reliability;
       K ∈ {1,2}, the number of modules for each type needed, equaling 1
         for not allocating backup while 2 for allocating backup;
Output: A Boolean flag denoting whether success or failure.
9.  m_{r,c}^{targ} = Null,∀r ∈ R, c = 1 ... K; //Initialize target modules.
10. m_{r,c}^{cur} = L_r[c], ∀r ∈ R, c = 1... K; //Initialize K current modules of each
    type to be the first K modules in each list.
11. While (m_{r,c}^{cur} ≠ Null,∀r ∈ R, ,c = 1 ... K)
12.   ℛ_{temp} = reliability of i if assigned with m_{r,c}^{cur}, ∀r, c, computed
      according to (1) if K = = 1; (3) if K = = 2;
13.   If ℛtemp ≥ θ_i //θ_i, reliability requirement of i
14.     m_{r,c}^{targ} = m_{r,c}^{cur},∀r ∈ R, c = 1 ... K;
15.     Among m_{r,c}^{cur}, ∀r ∈ R, c = 1 ... K, find the module with the
        highest reliability m_r*,_c*^{cur}, replace it with a subsequent module
        in L_r* that has not been checked;
16.   Else
17.   Break;
18. If (m_{r,c}^{targ} ≠ Null,∀r ∈ R, ,c = 1 ... K)
19.   Allocate resource to i from the found target modules;
20.   Return True;
21. Return False;
```

[0042] Algorithm 1 is executed to allocate resources to service request i. Algorithm 1 starts by excluding modules that have insufficient resources remaining with them (Line 2), e.g., the remaining resources are less than service request i's demand D_{ir} (r∈R). Then, for each resource type, a list L_r (r∈R) is created to store the modules of this resource type. Since there are |R| types of resources, the number of lists is also |R|. Each list L_r (r∈R) is sorted in descending order of the reliabilities of modules stored in the list (Line 3). In such an order, the former modules are more reliable than the latter, and if the former cannot satisfy the reliability requirement of the service request, there is no need to check subsequent modules further, and the service request can be immediately rejected. Of course, if the ones at the top are feasible, it is necessary to take a further step to find less reliable modules which are still adequately reliable to guarantee the reliability requirement of service request i. In Line 4, resources are attempted to be allocated to service request i with only working resources by invoking the function Trial( ) (with input K=1). If the reliability requirement can be satisfied, the function Trial( ) accepts the input and Algorithm 1 returns an integer 1 (Line 5). Otherwise, both working and backup resources are attempted to be allocated by invoking the same function with input K=2 (Line 6) and Algorithm 1 returns an integer 2 (Line7). If both trials fail, Algorithm 1 returns −1 (Line 8), indicating that this service request cannot be satisfied and is therefore rejected.

[0043] Line 9~21 in Algorithm 1 provides the pseudocode of the function Trial( ) which has three inputs, e.g., current service request i, lists of modules L_r of each resource type (∀r∈R), each of which is already sorted (Line 3), and an integer K∈{1,2}. When K=1, the function tries to allocate only working resources to service request i while meeting its reliability requirement. If K=2, the function tries to allocate both working and backup resources to service request i to meet its requirement. In Line 9, $m_{r,c}^{targ}$ represents the target modules that are finally allocated to service request i which are initialized as NULL. Although only one term $m_{r,c}^{targ}$ is used, it represents |R|×K variables. For example, if K=2 (corresponding to the situation that both working and backup resources are needed) and three types of modules are considered, this term represents six variables which are the final six chosen modules for i, e.g., $m_{\text{“}CPU\text{”},1}^{targ}$ denotes the first CPU module (or working module) that are finally allocated to the current service request i; $m_{\text{“}CPU\text{”},2}^{targ}$ denotes the second CPU module (or backup module) that are finally allocated to service request i, and $m_{\text{“}Memory\text{”},1}^{targ}$ denotes the first memory module that is finally allocated to service request i. In Line 10, the function initializes |R|×K temporary variables denoted by $m_{r,c}^{cur}$ be the first K modules in each list. For example, if K=1, the first module in each list is checked to find whether it can fulfill the reliability requirement of service request i (Line 12). Similarly, if K=2, the first two modules in each list are attempted, where one is the working module of service request i's working module, and the other is its backup module. In Line 12, based on the value of K the reliability model is chosen either from (1) (no backup situation) or (3) (with a backup situation) to calculate the reliability of service request i. If these temporarily chosen modules ($m_{r,c}^{cur}$) can meet the

requirement (Line 13), these modules are temporarily set as the final chosen module (Line 14). In Line 15, in each loop, among all $m_{r,c}^{cur}$, ∀r∈R, c=1 . . . K, only one module is changed with the highest reliability to the next module that has not been checked in the list where this module is stored. The loop terminates when no more modules can be found (Line 11) or when the current modules cannot meet service request i's requirement (Line 17). Finally, in the function Trial( ), resources are allocated to service request i (Line 19) and report a success (Line 20) if this process succeeds. Otherwise, a failure is reported (Line 21).

[0044] The example illustrated in FIG. **4** is used to further illustrate the embodiments of the heuristic process. In this example, it is assumed that K=1, e.g., only working resources and only one module of each type are attempted to be allocated. There are three types of modules, e.g., CPU, memory, and GPU modules, where each type of module is arranged in a distinct list in descending order of reliability (represented by the value in each box). Suppose the reliability requirement of the current service request is θ_i=0.965. In FIGS. **4**, p1, p2, and p3 are pointers that point to current modules (e.g., $m_{r,c}^{cur}$) in each list in the process. In the beginning, modules c1, m1, and g1 are selected, and according to (1), the achieved reliability is $\mathcal{R}_{temp}$=0.998×0.994×0.991=0.983>θ_i, meaning that these three modules can meet service request i's reliability requirement. Next, p1 is moved to c2 as c1 has the highest reliability among c1, m1, and g1. The reliability is now $\mathcal{R}_{temp}$=0.975>θ_i. The next step is to move p2 to m2 as m1 has the highest reliability, to get $\mathcal{R}_{temp}$=0.969>θ_i. The next step is to move p3 to g2. But this now gives $\mathcal{R}_{temp}$=0.963<θ_i, meaning that current module combinations cannot fulfill service request i's reliability. The process stops here, and the final modules selected are c2, m2, and g1.

[0045] The complexity of Algorithm 1 is analyzed in the following. Assume that the adopted sorting algorithm takes T(n) time, where n is the number of elements in a sequence to be sorted. Accordingly, Lines 1~3 take $O(|R|\cdot T(|M_r|))$ time, mainly for sorting modules of each type, where $|M_r|$ is the number of modules of each resource type r E R. The most time-consuming part of the Trial( ) function algorithm is the while-loop (Lines 11-17). Each line inside the loop-body is executed at most once in each loop. Accordingly, the while-loop takes $O(|R|\cdot|M_r|)$ time in the worst case. Other parts of the function take O(1) time. Therefore, both Line 4 and Line 6 take $O(|R|\cdot|M_r|)$ time to call the function Trial( ). In total, the complexity of Algorithm 1 is in the order of $O(|R|\cdot T(|M_r|)+|R|\cdot|M_r|)$.

[0046] Algorithm 1 allocates resources for only one service request. For the scenario of batch service requests, the order of service requests directly impacts the total number of acceptances. To maximize the objective function (4), a shuffling process is employed. For this, given a set of requests I, the order of requests in I is randomized, and then resources for each request according to the randomized order are attempted to be allocated by invoking Algorithm 1. This randomize-then-attempt procedure is executed many times and finally the result with the maximum value of the objective function (4) is chosen.

[0047] The heuristic process is executed each time a resource allocation is made to determine whether to accept or reject a service request. The embodiments of the heuristic process and their examples are described and illustrated above with only a static scenario where all the service requests are assumed to be known in advance. Nevertheless, the heuristic process is also applicable to a dynamic situation, and the execution frequency of this algorithm depends on the arrival rate of the service requests.

[0048] The functional units, modules, models, and algorithms of the apparatuses and the methods in accordance to the embodiments disclosed herein may be implemented using electronic devices, computer processors, or electronic circuitries including but not limited to application specific integrated circuits (ASIC), field programmable gate arrays (FPGA), and other programmable logic devices configured or programmed according to the teachings of the present disclosure. Machine instructions running in the electronic devices, computer processors, or programmable logic devices can readily be prepared by practitioners skilled in the software or electronic art based on the teachings of the present disclosure.

[0049] All or portions of the methods in accordance to the embodiments may be executed in one or more electronic devices including server computers, personal computers, laptop computers, mobile computing devices such as smartphones and tablet computers.

[0050] The embodiments include computer storage media having machine instructions stored therein which can be used to configured microprocessors to perform any of the processes of the present invention. The storage media can include, but are not limited to, floppy disks, optical discs, Blu-ray Disc, DVD, CD-ROMs, and magneto-optical disks, ROMs, RAMs, flash memory devices, or any type of media or devices suitable for storing instructions, codes, and/or data.

[0051] Each of the functional units in accordance to various embodiments also may be implemented in distributed computing environments and/or Cloud computing environments, wherein the whole or portions of machine instructions are executed in distributed fashion by one or more processing devices interconnected by a communication network, such as an intranet, Wide Area Network (WAN), Local Area Network (LAN), the Internet, and other forms of data transmission medium.

[0052] The foregoing description of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Many modifications and variations will be apparent to the practitioner skilled in the art.

[0053] The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention for various embodiments and with various modifications that are suited to the particular use contemplated.

What is claimed is:

1. An apparatus for resource allocation in a disaggregated data center (DDC), comprising:

a first processor configured to execute a reliability model to determine an achievable reliability for a service request to the DDC; and

a second processor configured to execute an integer linear programming (ILP) model to perform a resource allocation for the service request to the DDC;

wherein the DDC comprises a plurality of nodes, each of the nodes comprises one or more computing resources of only one computing resource type;

wherein the computing resource type is selected from a group of computing resource types comprising: central processing unit (CPU), graphical processing unit (GPU), transient memory circuitry, and non-transient memory circuitry;

wherein the DDC comprises multiple computing resource types, and the execution of the service requested received by the DDC requires performance of computing resource of at least one of the computing resource types; and

wherein nodes of same computing resource type are configured to form a parallel system such that as long as at least one of the nodes in the parallel system is available, the parallel system is available for performance in the execution of the service requested received by the DDC.

2. The apparatus of claim **1**, further comprising:

a third processor configured to execute a heuristic process to perform a resource allocation for the service request to the DDC.

3. The apparatus of claim **1**,

wherein the reliability model determines the achievable reliability by computing a product of reliabilities of all of the parallel systems;

wherein a reliability is a probability of normal working of a system or a node;

wherein each of the parallel systems comprises at least one working node $\mathcal{M}_r^W$ and one backup node $\mathcal{M}_r^B$ of computing resource type r;

wherein the working node $\mathcal{M}_r^W$ having a reliability $\mathcal{R}_{\mathcal{M}_r^W}$;

wherein the backup node $\mathcal{M}_r^B$ having a reliability $\mathcal{R}_{\mathcal{M}_r^B}$;

wherein the working node Wand the backup node $\mathcal{M}_r^B$ are arranged to form the parallel system of computing resource type r; and

wherein a reliability of the the parallel system of computing resource type r is obtained by computing:

$$1-(1-\mathcal{R}_{\mathcal{M}_r^W})\cdot(1-\mathcal{R}_{\mathcal{M}_r^B}).$$

4. The apparatus of claim **1**,

wherein each service request received by the DDC is allocated with at least one of one working computing resource and one backup computing resource for each computing resource type necessary for an execution of the service request;

wherein the performance of the ILP model comprises:

maximizing total number of service requests received by the DDC accepted for execution;

minimizing number of the accepted service requests allocated with backup computing resources; and

subjecting to one or more constraints comprising:

the service request is allocated with only one node for each computing resource type;

the working node and the backup node allocated to the service request do not share a same computing resource;

if a working computing resource is allocated to the service request, no backup computing resource is allocated to the service request, else if a backup computing resource is allocated to the service request, a working computing resource is also allocated to the service request;

if a working computing resource is allocated to the service request, the service request is accepted for execution regardless of whether a backup computing resource is allocated to the service request;

a total resource demand of all service requests allocated with a node and pending for execution is not higher than a resource capacity of the node; and

a reliability of computing resources of a computing resource type in the DDC must equal or higher than a reliability requirement of the service request before its acceptance for execution.

5. The apparatus of claim **2**, wherein the heuristic process comprises:

excluding one or more of the nodes for being available for allocation to a service request received by the DDC, wherein the nodes have insufficient resource capacity to satisfy a resource demand of the service request;

sorting the nodes of each of the computing resource types necessary for an execution of the service request based on a reliability of each of the nodes of the computing resource type into lists of nodes of the computing resource types necessary for the execution of the service request;

iterating through each of the lists to find a first node with a least reliability among the nodes of each of the computing resource types necessary for the execution of the service request such that a product of the reliabilities of the first nodes of all of the computing resource types necessary for the execution of the service request is equal or higher than a reliability requirement of the service request;

if the first nodes are found such that the product of the reliabilities of the first nodes is equal or higher than a reliability requirement of the service request, the first nodes found are allocated to the service request as its working nodes;

else if the first nodes are not found such that the product of the reliabilities of the first nodes is equal or higher than a reliability requirement of the service request, then iterating through each of the lists to find a first and second node pair with two least reliabilities among the nodes of each of the computing resource types necessary for the execution of the service request such that a product of the reliabilities of the first and second node pairs each acting as a parallel system of all of the computing resource types necessary for the execution of the service request is equal or higher than a reliability requirement of the service request.

6. A method for resource allocation in a disaggregated data center (DDC), comprising:

executing a reliability model to determine an achievable reliability for a service request to the DDC; and

executing an integer linear programming (ILP) model to perform a resource allocation for the service request to the DDC;

wherein the DDC comprises a plurality of nodes, each of the nodes comprises one or more computing resources of only one computing resource type;

wherein the computing resource type is selected from a group of computing resource types comprising: central processing unit (CPU), graphical processing unit (GPU), transient memory circuitry, and non-transient memory circuitry;

wherein the DDC comprises multiple computing resource types, and the execution of the service requested received by the DDC requires performance of computing resource of at least one of the computing resource types; and

wherein nodes of same computing resource type are configured to form a parallel system such that as long as at least one of the nodes in the parallel system is available, the parallel system is available for performance in the execution of the service requested received by the DDC.

7. The method of claim **6**, further comprising:

executing a heuristic process to perform a resource allocation for the service request to the DDC.

8. The method of claim **6**,

wherein the reliability model determines the achievable reliability by computing a product of reliabilities of all of the parallel systems;

wherein a reliability is a probability of normal working of a system or a node;

wherein each of the parallel systems comprises at least one working node $\mathcal{M}_r^W$ and one backup node $\mathcal{M}_r^B$ of computing resource type r;

wherein the working node $\mathcal{M}_r^W$ having a reliability $\mathcal{R}_{\mathcal{M}_r^W}$;

wherein the backup node $\mathcal{M}_r^B$ having a reliability $\mathcal{R}_{\mathcal{M}_r^B}$;

wherein the working node $\mathcal{M}_r^W$ and the backup node $\mathcal{M}_r^B$ are arranged to form the parallel system of computing resource type r; and

wherein a reliability of the the parallel system of computing resource type r is obtained by computing:

$$1-(1-\mathcal{R}_{\mathcal{M}_r^W})\cdot(1-\mathcal{R}_{\mathcal{M}_r^B}).$$

9. The method of claim **6**,

wherein each service request received by the DDC is allocated with at least one of one working computing resource and one backup computing resource for each computing resource type necessary for an execution of the service request;

wherein the performance of the ILP model comprises:

maximizing total number of service requests received by the DDC accepted for execution;

minimizing number of the accepted service requests allocated with backup computing resources; and

subjecting to one or more constraints comprising:

the service request is allocated with only one node for each computing resource type;

the working node and the backup node allocated to the service request do not share a same computing resource;

if a working computing resource is allocated to the service request, no backup computing resource is allocated to the service request, else if a backup computing resource is allocated to the service request, a working computing resource is also allocated to the service request;

if a working computing resource is allocated to the service request, the service request is accepted for execution regardless of whether a backup computing resource is allocated to the service request;

a total resource demand of all service requests allocated with a node and pending for execution is not higher than a resource capacity of the node; and

a reliability of computing resources of a computing resource type in the DDC must equal or higher than a reliability requirement of the service request before its acceptance for execution.

10. The method of claim **7**, wherein the heuristic process comprises:

excluding one or more of the nodes for being available for allocation to a service request received by the DDC, wherein the nodes have insufficient resource capacity to satisfy a resource demand of the service request;

sorting the nodes of each of the computing resource types necessary for an execution of the service request based on a reliability of each of the nodes of the computing resource type into lists of nodes of the computing resource types necessary for the execution of the service request;

iterating through each of the lists to find a first node with a least reliability among the nodes of each of the computing resource types necessary for the execution of the service request such that a product of the reliabilities of the first nodes of all of the computing resource types necessary for the execution of the service request is equal or higher than a reliability requirement of the service request;

if the first nodes are found such that the product of their reliabilities is equal or higher than a reliability requirement of the service request, the first nodes found are allocated to the service request as its working nodes;

else if the first nodes are not found such that the product of the reliabilities of the first nodes is equal or higher than a reliability requirement of the service request, then iterating through each of the lists to find a first and second node pair with two least reliabilities among the nodes of each of the computing resource types necessary for the execution of the service request such that a product of the reliabilities of the first and second node pairs, each pair acting as a parallel system, of all of the computing resource types necessary for the execution of the service request is equal or higher than a reliability requirement of the service request; and

if the first and second node pairs are found such that the product of their reliabilities, each first and second node pair acting as a parallel system, is equal or higher than a reliability requirement of the service request, the first nodes found are allocated to the service request as its working nodes and the second nodes found are allocated to the service request as its working nodes.

* * * * *