ISSN 1064-2307, Journal of Computer and Systems Sciences International, 2014, Vol. 53, No. 6, pp. 854–866. © Pleiades Publishing, Ltd., 2014.
Original Russian Text © P.M. Vdovin, V.A. Kostenko, 2014, published in Izvestiya Akademii Nauk. Teoriya i Sistemy Upravleniya, 2014, No. 6, pp. 80–93.

COMPUTER METHODS

Algorithm for Resource Allocation in Data Centers with Independent Schedulers for Different Types of Resources

P. M. Vdovin and V. A. Kostenko
Moscow State University, Moscow, Russia
e-mail: Pavel.vdovin@gmail.com
Received April 3, 2014; in final form, May 14, 2014

Abstract—An algorithm for assigning requests to physical recourses for data centers with independent schedulers for different types of resources (computational resources, network resources, and data storages) is considered. This algorithm is based on the combination of greedy and limited search strategies. The algorithm provides a required balance between the computational complexity and quality of assignments by limiting the maximum allowed search depth. Theoretical and experimental results of investigating its features are presented.

DOI: 10.1134/S1064230714050141

INTRODUCTION

In [1, 2], a mathematical model of a data center (DC) was developed that makes it possible to describe a wide class of DC architectures as well as the mathematical statement of the resource allocation problem in the Infrastructure-as-a-Service (IaaS) mode [3] is formulated. In the proposed model, computational resources, data storages, and network resources are considered as scheduled types of resources and are scheduled consistently in the sense of compliance with the service-level agreement (SLA) [4]. This allows one to specify SLA for all types of DC resources and to construct maps of requests onto physical resources for which the required SLAs are guaranteed.

In [2], various approaches to resource allocation in the DC are compared. In this paper, we consider an algorithm of assigning requests to physical resources for DCs with independent schedulers for different types of resources and present the results of investigating its features.

Problems of scheduling in real-time systems [5–10] are viewed as the closest problems in terms of the assured compliance with the SLA. In these problems, the constraints placed on the feasible time of executing applied programs or program processes are used as the SLA criteria that cannot be violated. In the problem under consideration, the requirements for the "volume" of physical recourses allocated to request elements are used as the SLA criteria. This makes it difficult to use the available scheduling algorithms for this problem.

1. MATHEMATICAL FORMULATION OF THE RESOURCE ALLOCATION PROBLEM IN DATA CENTERS

The model of DC physical resources is described by the graph $H = (P \cup M \cup K, L)$, where P is the set of computation nodes, M is the set of data storages, K is the set of commutation elements of the DC data communication network, and L is the set of physical data transmission channels. On the sets P, M, K, and L, vector functions are defined whose arguments are elements of the sets P, M, K, or L, respectively. The values of these functions are the characteristics of the corresponding computation node, data storage, commutation element, or data transmission channel:

$$(ph_1, ph_2, \ldots, ph_{n1}) = fph(p), \quad p \in P,$$

$$(mh_1, mh_2, \ldots, mh_{n2}) = fmh(m), \quad m \in M,$$

$$(kh_1, kh_2, \ldots, kh_{n3}) = fkh(k), \quad k \in K,$$

$$(lh_1, lh_2, \ldots, lh_{n4}) = flh(l), \quad l \in L.$$

In this work, we assume that the commutation elements and data transmission channels have the same characteristics. Mostly, the following characteristics of physical elements are used in practice:

(1) for computation nodes: ⟨number of cores⟩, ⟨frequency⟩, ⟨core type⟩, ⟨RAM capacity⟩, and ⟨disk memory capacity⟩;

(2) for data storages: ⟨storage capacity⟩ and ⟨data storage type⟩;

(3) for commutation elements: ⟨bandwidth⟩ and ⟨delay⟩;

(4) for data transmission channels: ⟨bandwidth⟩ and ⟨delay⟩.

A resource request is specified by the graph $G = (W \cup S, E)$, where W is the set of virtual machines used by applications, S is the set of storage elements, and E is the set of virtual links between the virtual machines and the storage elements of the request. On the sets W, S, and E, vector functions are defined whose arguments are elements of the sets W, S, and E, respectively. The values of the functions are characteristics (the required quality of service (SLA)) of the corresponding virtual machine, storage element, or virtual link:

$$(wg_1, wg_2, \ldots, wg_{n1}) = fwg(w), \quad w \in W,$$

$$(sg_1, sg_2, \ldots, sg_{n2}) = fsg(s), \quad s \in S,$$

$$(eg_1, eg_2, \ldots, eg_{n4}) = feg(e), \quad e \in E.$$

The characteristics of an SLA request element are the same as the characteristics of the physical resource corresponding to this element.

By the resource request assignment, we mean a map

$$A : G \to H = \{W \to P, S \to M, E \to \{K, L\}\}.$$

Let us describe three types of relations between the request characteristics and the corresponding physical resource characteristics. We denote the characteristic of the ith request by x_i and the corresponding characteristic of the jth physical resource by y_j. Then, these relations can be written as follows:

(1) The physical resource capacity must not be overloaded:

$$\sum_{i \in R_j} x_i \le y_j,$$

where R_j is the set of requests assigned to be executed on the physical resource j and x_i is the corresponding characteristic of the assigned request.

(2) The type of the requested resource must match the type of the physical resource:

$$x_i = y_j.$$

(3) The physical resource must possess the required characteristics:

$$x_i \le y_j.$$

The *assignment* $A : G \to H = \{W \to P, S \to M, E \to \{K, L\}\}$ is called correct if relations 1–3 hold for all physical resources and all their characteristics.

The *residual graph* of available resources is the graph H_{res} for which the values of the functions are redefined according to the corresponding characteristics that must satisfy relation (1):

$$\begin{aligned} fph_{res}(p) &= fph(p) - \sum_{w \in W_p} fwg(w), \\ fmh_{res}(m) &= fmh(m) - \sum_{s \in S_m} fsg(s), \\ fkh_{res}(k) &= fkh(k) - \sum_{e \in E_l} feg(e), \quad flh_{res}(l) = flh(l) - \sum_{e \in E_k} feg(e). \end{aligned} \tag{1.1}$$

Here, W_p is the set of virtual machines assigned to be executed on the computation node p, E_l is the set of virtual links mapped onto the physical channel l, E_k is the set of virtual links passing through the commutation element k, and S_m is the set of storage elements in the data storage m.

The input of the problem of assigning physical resources to resource requests is as follows:

(1) the set of requests $Z = \{G_i\}$ received by the scheduler;

(2) the residual graph of available resources $H_{res} = (P \cup M \cup K, L)$.

It is required to assign the maximum number of requests from the set Z to the DC such that the assignment A is correct.

In the case when the virtual data storage is failed to be assigned, the replication procedure may be called that duplicates the data on several physical data storages. This problem arises if we have virtual data storages with high intensity of reading and low intensity of writing. In this case, no high bandwidth channel is needed to ensure data consistency; some of the applications can work with a virtual data storage and the other applications may work with its copy placed on another physical data storage.

By replication, we mean a map $R : H \to H$ that duplicates the data of a certain $m \in M$ and creates a virtual link for maintaining the data consistency $(m', l_1, k_1, \ldots, k_{n-1}, l_n, m)$, where $k_i \in K$, $l_i \in L$, $m' \in M$; m' is a replica of the data storage m.

If a storage element is a replica of a certain virtual data storage s, then the node s' is added to the graph of requested resources G. The virtual link between the nodes s and s' is also added to the graph G; the bandwidth of this link is determined based on the requirement of ensuring the consistency of the replica and the original database.

The *input of the algorithm for assigning requests to physical resources* consists of the residual graph of available resources H_{res} and the set of resource requests $\{G_i\}$. The set $\{G_i\}$ is formed by the task manager. In addition to the newly received requests, this set can also include requests that are already executed and for which migration is acceptable. The task manager also determines the starting time of the scheduler.

The *output of the algorithm for assigning requests to physical resources* consists of the set of resource request assignments $\{A_i : G_i \to H, i = \overline{1,n}\}$ and the set of replications $\{R_i\}, i = 0,1,\ldots$.

2. ALGORITHMS FOR ASSIGNING REQUESTS TO DATA CENTER PHYSICAL RESOURCES

The algorithm for assigning requests to physical resources using independent schedulers involves the three following steps:

S t e p 1. Assigning virtual machines to computation nodes;

S t e p 2. Assigning storage elements to physical data storages;

S t e p 3. Mapping request virtual links onto the physical data transmission channels and commutation elements for the assignments of the virtual machines and storage elements to physical resources obtained at Steps 1 and 2.

2.1. Assignment of Virtual Machines and Storage Elements

In scheduling theory, problems of assigning physical resources to virtual machines and storage elements are known as bin packing problems [11]. The proposed algorithm is based on a combination of greedy and limited search strategies. It assigns request elements according to the greedy scheme; in the case when the next element is failed to be assigned, the algorithm calls the limited search procedure. For this procedure, the parameter is specified that limits the search depth. This makes it possible to set a required balance between the computational complexity and accuracy of the algorithm.

The general scheme of the procedure for request element assignment (virtual machines or storage elements) is as follows.

S t e p 1. Select the next request G_i from the set of resource requests $\{G_i\}$ using the greedy criterion K_G.

S t e p 2. Select the next element e (virtual machine $e \in W, W \in G_i$, or storage element $e \in S, S \in G_i$) using the greedy criterion K_e.

S t e p 3. Form the set of physical resources Ph ($Ph \subseteq P$ or $Ph \subseteq M$, respectively) to which the selected element e can be assigned, i.e., for which the assignment the request e to the physical resource is correct.

S t e p 3.1. If the set Ph is empty, then call the limited search procedure. If this procedure fails, then the request G_i is failed to be assigned; in this case, eliminate the previously assigned elements of the request G_i and redefine the values of the physical resource characteristics using functions (1.1). If the set $\{G_i\}$ is not empty, go to Step 1; otherwise, terminate the algorithm.

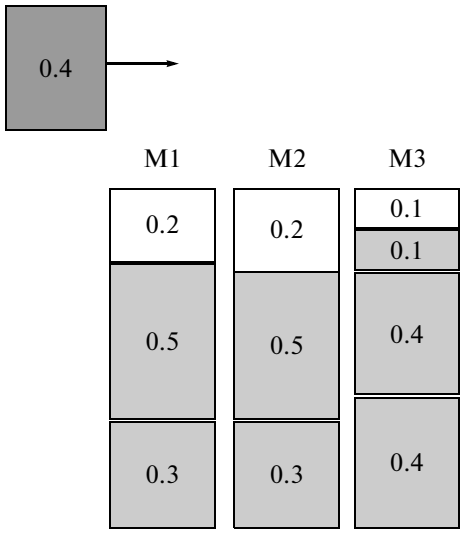

Fig. 1. Example of the limited search procedure execution.

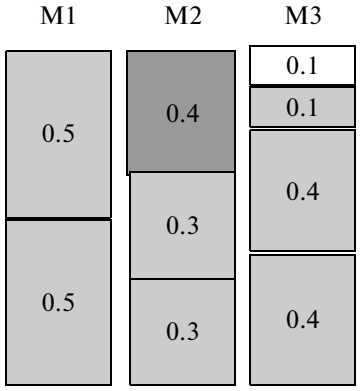

Fig. 2. Reassignment of elements from Fig. 1.

S t e p 3.2. If the set Ph is not empty, then select the physical element $ph \in Ph$ to be assigned according to the greedy criterion K_{ph} and assign the request element e to the physical element ph; redefine the values of its characteristics using functions (1.1). If there are unconsidered request elements, go to Step 2. If the set $\{G_i\}$ is not empty, go to Step 1; otherwise, terminate the algorithm.

2.1.1. Limited search procedure. The limited search procedure is called when the next request element e ($e \in W$ or $e \in S$) fails to be assigned to any physical resource ph ($ph \in P$ or $ph \in M$, respectively). The principle of operation of the limited search procedure is illustrated in Figs. 1 and 2. Figure 1 shows a situation in which a virtual element with capacity 0.4 fails to be assigned to any physical element despite the fact that the total free capacity is sufficient for the element to be assigned. By reassigning the elements as shown in Fig. 2, the fragmentation is eliminated and the virtual element can be assigned.

The limited search is limited by the given number m that specifies the maximum number of physical elements for which the reassignment is allowed (for $m = 2$, the assigned elements can be withdrawn and reassigned from not more than two physical elements at a time).

The general scheme of the limited search procedure is as follows.

S t e p 1. For the given m, form all sets $A_m \subseteq A$ consisting of not more than m physical resources and search according to the decrease of the total value calculated using the greedy criterion K_G.

S t e p 1.1. If, for a certain set A_m, the total residual capacity of resources is sufficient to assign the current element, then execute Steps 1.1.1–1.1.3 of the algorithm. Otherwise, go to Step 2.

S t e p 1.1.1. Withdraw the elements that are assigned to physical resources from the set A_m.

S t e p 1.1.2. Try to perform reassignment using one of the following techniques:

(a) first, assign the element e to a physical resource from the set A_m using the greedy strategies and then assign all the virtual elements withdrawn from the physical elements A_m using the greedy strategies;

(b) form the set consisting of the element e and all the virtual elements withdrawn from the physical elements of A_m; then assign the formed set according to the greedy strategies (see Steps 2 and 3 of the general scheme for assigning request elements without limited search);

(c) form the set consisting of the element e and all the virtual elements withdrawn from the physical elements of A_m; perform the exhaustive search for feasible assignments of the elements from the formed set.

S t e p 1.1.3. If the reassignment fails, then return the initial assignments of elements.

S t e p 2. If the element e fails to be assigned upon searching all the above-mentioned sets, then return failure.

2.1.2. Greedy criteria. The quality of results produced by the algorithm considerably depends on the greedy criteria used. The criteria are specified at Step 1 of the general scheme of the request element assignment procedure when the request to be assigned is selected (K_G), at Step 2 when the request element is selected (K_e), and at Step 3 when the physical resource to which the element is assigned (K_{ph}) is selected.

The proposed criteria are based on the cost function $r(e)$ of assigning the element e. We propose two techniques to calculate the cost of the element assignment:

(1) dynamic selection of the resource characteristic that is the most loaded at the time of assignment and the calculation of cost based on this characteristic;

(2) calculation of the weighted sum of resource characteristics with respect to their workload.

Let an element be characterized by the vector of required resource characteristics ($r_{e,1}, r_{e,2}, \ldots r_{e,n}$) for which the overload is not allowed.[1] In the case of dynamic selection, the most loaded characteristic is determined prior to each assignment:

$$max\_resource = \arg\max_i \left(\frac{\sum\limits_{G_a} \sum\limits_{e \in E,\, E \in G_a} r_{e,i}}{\sum\limits_{ph \in Ph} r_{ph,i}} \right).$$

In this formula, the summation in the numerator is performed over all the assigned requests G_a. Thus, the workload of a resource is calculated as the quotient from the division of the total capacity of the assigned elements by the total capacity of the physical elements for the given characteristic.

In the case of dynamic selection, the cost function of an element is calculated as the capacity of the most loaded characteristic: $r(e) = r_{e,max\_resource}$.

To determine the weighted cost of an element, the deficit is calculated for each resource:

$$d(i) = \frac{\sum\limits_{G} \sum\limits_{e \in E,\, E \in G} r_{e,i}}{\sum\limits_{ph \in Ph} r_{ph,i}}.$$

This quantity is determined as the quotient in which the numerator is the total required capacity for the given characteristic for all the requests G and the denominator is the total value of the available physical resources. The greater the deficit, the greater is the required amount of resources of this type.

The cost function is calculated as[2]

$$r(e) = \sum_{i=1..n} d(i)\, r_{e,i};$$

it is the weighted sum of required resources with regard to their deficit.

[1] In this case, we consider only the characteristics for which the first relation between the request characteristics and the corresponding physical resource characteristics is verified; i.e., the total capacity of the virtual elements assigned to the resource must not exceed the capacity of the physical resource; by the time of using the criteria, the filtration of the elements has already been performed by the two other relations.

[2] In this formula, the value of resource capacity is assumed to be normalized.

There are two types of criteria: compact assignment and balanced assignment.

The compact assignment is used when the physical resources of the network are not critical resources, i.e., when the ratio of the costs of the required network resources to the available network resources is below a given threshold. For the compact assignment, we propose the following criteria:

(1) K_G: select the request G with the highest total required cost of elements;

(2) K_e: among elements of the request G, select the element e that has the maximum required capacity $r_v(e)$;

(3) K_{ph}: among the physical elements $ph \in Ph$ that have the sufficient amount of resources to assign the selected element e, select the element with the lowest cost (that is calculated similarly to the cost of a virtual element).

The greedy criteria described above provide the assignment using the best-fit decreased (BFD) strategies [11]. In the case of the one-dimensional bin packing problem, this strategy reaches the asymptotic accuracy of 11/9. This means that, in the worst case, this algorithm assigns elements to the number of physical resources that is 11/9 times greater compared with the optimal algorithm.

The balanced assignment is used when the network resources are critical, i.e., when the ratio of the cost of the required network resources to the cost of the available network resources exceeds a certain threshold.

In this case, the first two criteria (K_G and K_e) correspond to the compact assignment. For the criterion K_{ph}, we propose to select the element with the highest cost from the set of elements $ph \in Ph$ that have a sufficient capacity for the selected element e to be assigned. This approach allows one to distribute virtual resources over all available physical elements while ensuring a greater number of network resources for the virtual channel assignment.

2.2. Assigning Virtual Links to Physical Resources of the Network

The assignment of virtual links is performed upon obtaining the virtual machines and storage elements are assigned; it is based on solving the problem of finding the k shortest paths in the graph [12]. If a certain virtual channel $e \in E$ fails to be assigned, then the limited search procedure is called that searches through the sets with limited cardinality of already assigned virtual links in order to reassign them so that the given virtual link could be assigned. The general scheme of the limited search for virtual channels is similar to the one for virtual machines and storage elements. In the case of failure, the replication procedure is called that creates a replica for the storage element to eliminate the overload of exchange channels in the physical data storage. When creating the replica, it is also required to organize a link for maintaining the consistency between the storage element and its replica.

Note that the replication procedure is used in this algorithm to optimize the allocation of virtual links, i.e., to create extra paths when assigning requests.

2.2.1. General scheme of virtual link assignment. The general scheme of the algorithm for assigning virtual links is as follows.

S t e p 1. Select the request $G \in G_A$ for the virtual link assignment according to the criterion K_G, where G_A is the set of requests for which the virtual machines and storage elements are already assigned.

S t e p 2. Select the virtual link $e \in E$ of the request G according to the criterion K_e.

S t e p 3. Assign the virtual link e using the algorithm for finding the k shortest paths and redefine the values of the physical resource characteristics using functions (1.1).

S t e p 4. In the case of failure, call the limited search procedure.

S t e p 5. If the limited search procedure returns failure, call the replication procedure.

S t e p 6. In the case of failure, the request G cannot be assigned: withdraw all the allocated virtual links, virtual machines, and storage elements of the request G and redefine the values of the physical resource characteristics using functions (1.1).

2.2.2. Greedy criteria. At the first two steps, we select criteria similar to those for allocating virtual machines and storage elements. Thus, it is proposed to select the request with the highest total cost of virtual links (K_G) and then sequentially select the virtual link with the maximum capacity (K_e). The cost is determined similarly to the cost of the computation resources/virtual machines and data storages/storage elements. Typically, only one characteristic—bandwidth—is used for data transmission channels and switches. In this case, the cost is calculated as the capacity value of this characteristic.

2.2.3. Assigning a virtual link. A virtual link is assigned using the algorithm for finding the k shortest paths (for example, the Yen's algorithm [12]). The general scheme of the algorithm is as follows.

S t e p 1. For the given k, search for not more than k shortest paths $p \in P$ of minimum length (by the length, we mean the number of switches in the path $p \in P$). Search only for the paths wherein each element possesses the sufficient capacity for the given virtual link to be assigned. If no paths are found, return failure.

S t e p 2. For the found paths, select the path $p \in P$ with the highest residual total cost among the data transmission channels and switches.

This search procedure employs two criteria: minimum length and maximum residual cost. The minimum length is the primary characteristic, since DC topologies (such as the FatTree topology [2]) have a tree-like structure (with redundant edges added) that allows one to select paths from the set of paths of the same length for each pair of elements.

2.2.4. Limited search procedure. The limited search procedure consists of the following steps.

S t e p 1. For the given m, search through all the sets $A_e \subseteq A$ consisting of not more than m allocated virtual links in descending order of the total cost of virtual links. For each set A_e, perform Steps 1.1–1.3.

S t e p 1.1. Withdraw all the selected virtual links $a \in A_e$.

S t e p 1.2. Try to assign the virtual link e using the algorithm for finding the k shortest paths; in the case of failure, restore the allocation of the withdrawn virtual links $a \in A_e$ and go to the next set $A_e \subseteq A$.

S t e p 1.3. Try to assign all the withdrawn virtual links $a \in A_e$; in the case of success, return success. Otherwise, withdraw the virtual link e and restore the initial assignments.

S t e p 2. If the virtual link e fails to be assigned upon searching through all the given sets $A_e \subseteq A$, then return failure.

2.2.5. Replication procedure. The replication is possible when one of elements connected by the virtual link $e = (w, s)$ is the storage element $s \in S$ for which the replication is available. The replication procedure consists in searching for the data storage $m \in M$ that has sufficient resources for assigning the replica (the replica requires as much resources as the given storage element s), and the total length of all virtual links $e = (w \in W, s)$ (that include the given storage element s) to this replica of m is minimum. In addition, it is required to create a link l for maintaining the consistency between the replica and the original storage element (the required bandwidth of the link l is determined by the type of the storage element s). If the link l for maintaining the consistency fails to be created, then another data storage s is considered in order of descending total path length of virtual links.

If the replication procedure returns success, then the virtual links that include the given storage element s can further be assigned to this replica of m. If the data storage $m \in M$ with a sufficient number of resources is not found or the link l for maintaining the consistency fails to be created, then the procedure returns failure.

3. COMPUTATIONAL COMPLEXITY OF THE ALGORITHM

Let N be the number of computation nodes, S be the number of data storages, and W be the number of switches in the data communication network. Let RV be the total number of virtual machines for all resource requests, RS be the total number of storage elements, and RVL be the total number of virtual links. We denote the search coefficient for the limited search of virtual machines and storage elements by K_{nodes}, the search coefficient for the assignment of virtual links by K_{links}, and the number of shortest paths in the procedure for finding the k shortest paths by K_{path}.

1. Assigning virtual machines to computation nodes. At this stage, the following operations are performed.

1.1. Sorting virtual machines by the chosen criterion; the complexity of this step is $O(RV \log(RV))$.

1.2. For each virtual machine, select a computation node to which the assignment is performed (if feasible); in the worst case, the complexity is $O(RV\,N)$ operations.

1.3. The limited search procedure. The number of combinations being searched is $C_N^{K_{\text{nodes}}}$ (N choose K_{nodes}). For each combination, the reassignment is attempted; the complexity of reassignment depends on the chosen technique of limited search:

(a) when using the greedy algorithm in the limited search procedure, the complexity is $O(RV(\log(RV) + N))$;

(b) when using the exhaustive search algorithm, the complexity is as high as $O(RV!N)$ in the worst case. At this step, the worst complexity is $O(RV(\log(RV)+N) + O(C_{RV}^{K_{\text{nodes}}} RV(\log(RV)+N)))$ when using the greedy algorithm in the limited search procedure and is $O(RV(\log(RV)+N) + O(C_{RV}^{K_{\text{nodes}}} RV!N))$ when using the exhaustive search of all possible maps in the limited search procedure.

2. Assigning storage elements to data storages. This stage is similar to the previous one and has the same complexity, but the number of storages is used in estimates rather than the number of virtual machines.

3. Assigning virtual links. At this stage, the following operations are performed.

3.1. Sort the virtual links by the chosen criterion; the complexity of this step is $O(RVL\log(RVL))$.

3.2. Try to assign the given virtual link using the procedure for finding the k shortest paths. This procedure consists uses Dijkstra's algorithm not more than $K_{\text{path}}(W+2)$ times and involves not more than $O(K_{\text{path}}(W+2)^2)$ operations (since the virtual machines and storage elements are already assigned, the number of nodes in the search graph is $W+2$). Thus, the complexity of this step is $O(RVL\,K_{\text{path}}(W+2)^3)$.

3.3. The limited search procedure of depth K_{links} for virtual links. The complexity of this procedure is estimated similarly to the virtual machine assignment; the only difference is in using the procedure for finding the k shortest paths. Thus, the complexity of this step is $O(C_{RVL}^{K_{\text{links}}}(K_{\text{links}}+1)K_{\text{path}}(W+2)^3)$, where $C_{RVL}^{K_{\text{links}}}$ is the number of combinations being searched and $K_{\text{links}}+1$ is the number of virtual links being reassigned.

3.4. The replication procedure. In the worst case, the replication is performed for each storage element. The complexity of searching for the data storage to be replicated is $O(S(N+W+1)^2)$; it finds the shortest paths to each data storage for a graph of $(N+W+1)$ elements. The complexity of finding the path for the link of maintaining the consistency is $O((W+2)^2)$. Thus, the overall complexity is $O(RVL(S(N+W+1)^2+(W+2)^2))$.

3.1. Dependence of the Algorithm Complexity on the Depth of Limited Search

When investigating the complexity of the algorithm with the zero depth of limited search (for $K_{\text{nodes}}=0$ and $K_{\text{links}}=0$), the following conclusions can be made:

(1) the complexity of the algorithm has a cubic dependence on W, quadratic dependence on N, and linear dependence on S (if the replication is not used, the algorithm also linearly depends on N);

(2) with increasing RV, RS, and RVL, the algorithm complexity increases as $O(RV\log RV)$, $O(RS\log RS)$, and $O(RVL\log RVL)$, respectively;

(3) the algorithm linearly depends on the parameter K_{path}.

When the search depth K_{nodes} and K_{links} increases, the algorithm complexity increases as $C_N^{K_{\text{nodes}}}(C_S^{K_{\text{nodes}}})$ and $C_{RVL}^{K_{\text{links}}}$, respectively. Note that the coefficient in the summands $O(C_N^{K_{\text{nodes}}} RV(\log(RV)+N))$ and $O(C_S^{K_{\text{nodes}}} RS(\log(RS)+S))$ increases with increasing K_{nodes}; the summands increase as $RV\log RV$ ($RS\log RS$, respectively) depending on RV (RS). The variation of the search complexity depending on K_{nodes} when using the greedy algorithm in the limited search procedure is shown in Table 1. In the case of using the exhaustive search in the limited search procedure, the complexity is increased by the factors of $(RV-1)!$ and $(RS-1)!$, respectively. The variation of complexity for the reassignment with exhaustive search is shown in Table 2.

However, when the value of K_{links} increases, the coefficient in the summand $O(C_{RVL}^{K_{\text{links}}}(K_{\text{links}}+1)K_{\text{path}}(W+2)^3)$, which has a cubic dependence on W, is changed. The variation of the limited search complexity depending on K_{links} is given in Table 3.

Table 1. Dependence of the algorithm complexity on the search depth K_{nodes} when using the greedy algorithm in the limited search procedure

Search depth K_{nodes}	Algorithm complexity coefficients depending on	
	N	S
1	N	S
2	$N(N-1)/2$	$S(S-1)/2$
3	$N(N-1)(N-2)/6$	$S(S-1)(S-2)/6$
...	...	...
K_{nodes}	$C_N^{K_{\text{nodes}}}$	$C_S^{K_{\text{nodes}}}$

Table 2. Dependence of the algorithm complexity on the search depth K_{nodes} when using the exhaustive search in the limited search procedure

Search depth K_{nodes}	Algorithm complexity coefficients depending on	
	N and RV	S and RS
1	$(RV-1)!N$	$(RS-1)!S$
2	$(RV-1)!N(N-1)/2$	$(RS-1)!S(S-1)/2$
3	$(RV-1)!N(N-1)(N-2)/2$	$(RS-1)!S(S-1)(S-2)/6$
...	...	...
K_{nodes}	$(RV-1)!C_N^{K_{\text{nodes}}}$	$(RS-1)!C_S^{K_{\text{nodes}}}$

Table 3. Dependence of the algorithm complexity on the search depth K_{links}

Search depth K_{links}	Algorithm complexity coefficients depending on
1	RVL
2	$RVL(RVL-1)/2$
3	$RVL(RVL-1)(RVL-2)/6$
...	...
N	$C_{RVL}^{K_{\text{links}}}$

4. EXPERIMENTAL EXPLORATION OF THE ALGORITHM PROPERTIES

The experimental exploration is performed to elucidate the following properties of the algorithm.

1. The dependence of the assignment quality and computational complexity of the algorithm on the chosen technique for calculating the cost of an element: the dynamic calculation with the selection of a critical characteristic or the weighted sum over all the criteria (see Subsection 2.1.2).

2. The dependence of the assignment quality and computational complexity of the algorithm on the strategies used in the limited search procedure: the greedy algorithm (when the current virtual element is assigned first or when the element is assigned together with the withdrawn elements) or the exhaustive search (see Subsection 2.1.1).

3. The dependence of the assignment quality and computational complexity of the algorithm on the assignment method: the compact or balanced assignment (see Subsection 2.1.2).

4. The efficiency of the replication procedure.

4.1. The Dependence on the Technique Used to Calculate the Cost of an Element

To investigate this dependence, a series of runs was performed for the randomly generated DC that consists only of computation nodes with the following characteristics:

(1) the number of computation nodes was 1000;

Table 4. Results of investigating the cost calculation technique for the full load

Cost calculation technique	Accuracy: average number of assigned requests	Average load of elements[5], %	Average execution time of algorithms, s
Dynamical with the use of the critical characteristic	95.63	96.3	63
Weighted sum with regard to resource deficit	93.77	97.7	34

[5] The average load is calculated as an arithmetical mean of the loads over all characteristics. This quantity is used to compare the packing densities.

Table 5. Results of investigating the cost calculation technique for the unbalanced load

Cost calculation technique	Accuracy: average number of assigned requests	Average load of elements	Maximum load over characteristics	Minimum load over characteristics	Average execution time of algorithms, s
		%			
Dynamical with the use of the critical characteristic	96.93	56.6	98.6	10	30
Weighted sum with regard to resource deficit	97.58	56.8	98.9	10	21

(2) the number of requests was 100 (on the average, 50 virtual machines for each request);

(3) the number of characteristics was four: ⟨number of cores⟩, ⟨frequency⟩, ⟨RAM capacity⟩, and ⟨disk memory capacity⟩; no overload of the physical resource for each characteristic was allowed.

(4) the tests were formed so that all the request could be assigned;

(5) the overall resource load (the ratio of the total required capacity of virtual elements to the capacity of physical elements over the given characteristic) was set using two methods:

the full load: the load over all characteristics was 100%;

the unbalanced load: ⟨number of cores⟩ was 10% load, ⟨frequency⟩ was 40% load, ⟨RAM capacity⟩ was 70% load, and ⟨disk memory capacity⟩ was 100% load.

For each combination *element cost calculation technique–resource load*, 100 runs were performed and the results were averaged. The limited search with greedy reassignment and the search depth of 1 was used.

The results depending on the cost calculation technique for the full and unbalanced load are summarized in Tables 4 and 5, respectively.

The results show that the dynamic cost calculation with the use of the critical characteristic assigns, on the average, a greater number of requests in the case of the full load; in the case of unbalanced load, the weighted sum assigns a greater number of requests. When the weighted sum is used to calculate the cost, the load of elements proves to be higher in both the cases.

4.2. The Dependence on the Chosen Procedure of Limited Search

Test cases were generated in the same way as in Subsection 4.1. The following versions of the limited search procedure were investigated:

without limited search;

with the greedy algorithm in the limited search procedure (the current virtual element was assigned first with the search depth equal to one);

with the greedy algorithm in the limited search procedure (the current virtual element was assigned first with the search depth equal to two);

with the greedy algorithm in the limited search procedure (the current virtual element was assigned together with all the withdrawn elements according to their cost);

with the exhaustive search in the limited search procedure.[3]

[3] In the case of exhaustive search and greedy algorithm with the search depth set to 2, an additional constraint was placed on the search procedure execution time: if there were no assignments made in the course of this time, then the procedure returned failure.

Table 6. Results of investigating the limited search procedure for the full load

Limited search technique	Accuracy: average number of assigned requests	Average load of elements, %	Average execution time of algorithms, s
Without limited search	93.22	97.4	7.5
Greedy algorithm: the current element is assigned first with the search depth 1	93.77	97.7	34
Greedy algorithm: the current element is assigned first with the search depth 2	93.9	97.7	230
Greedy algorithm: the current element is assigned together with all withdrawn elements	93.78	97.7	40
Exhaustive search	93.5	97.5	210

Table 7. Results of investigating the limited search procedure for the unbalanced load

Cost calculation technique	Accuracy: average number of assigned requests	Average load of elements	Maximum load over characteristics	Minimum load over characteristics	Average execution time of algorithms, s
		%			
Without limited search	97.09	56.6	98.6	10	9.8
Greedy algorithm: the current element is assigned first with search depth 1	97.58	56.8	98.9	10	21
Greedy algorithm: the current element is assigned first with search depth 2	97.7	56.8	99	10	95
Greedy algorithm: the current element is assigned together with all withdrawn elements	97.48	56.7	98.9	10	22
Exhaustive search	97.3	56.6	98.7	10	96

In all the cases, the cost function was calculated as the weighted sum over all the characteristics with regard to their deficit. The results for the full and unbalanced load are given in Tables 6 and 7, respectively.

It can be seen that different versions of limited search do not provide considerable improvement. The increase of the search depth also provides no considerable gain. This can be explained by the fact that independent requests were generated, and the limited search offers no advantages for such requests within reasonable time. As compared with the run with the limited search off, the average operating time for the unit search depth was increased more than twice. When the search depth is set to two or the exhaustive search was used, the time was increased by an order of magnitude.

Note that, in the case of unbalanced load, the algorithm shows better results (a high average number of assigned requests) in less time. This can be explained by the fact that the limited search procedure is called only when the element fails to be assigned, i.e., when physical elements are already heavily loaded. In the case of full load, this happens more often than in the case of unbalanced load, so the limited search procedure is called more frequently.

The use of the exhaustive search algorithm when the execution time of each search procedure is limited offers no advantages. In the case when there is no constraint placed on the execution time of the procedure, the time of each algorithm run is increased by several orders of magnitude and, therefore, is not included in the results.

4.3. The Dependence on the Assignment Method

Here, the criterion for selecting the physical elements when assigning virtual elements is investigated:

(1) the compact assignment: the physical element with the lowest cost is selected;

(2) the balanced assignment: the element with the highest cost is selected.

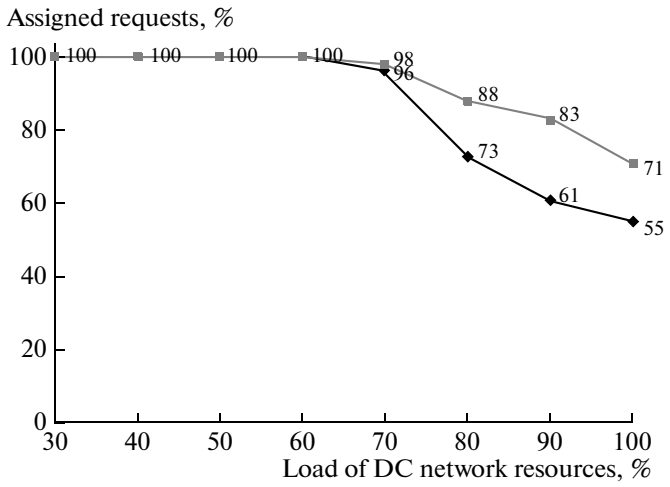

Fig. 3. Percentage of assigned requests for the compact (–◆–) and balanced (–■–) assignment.

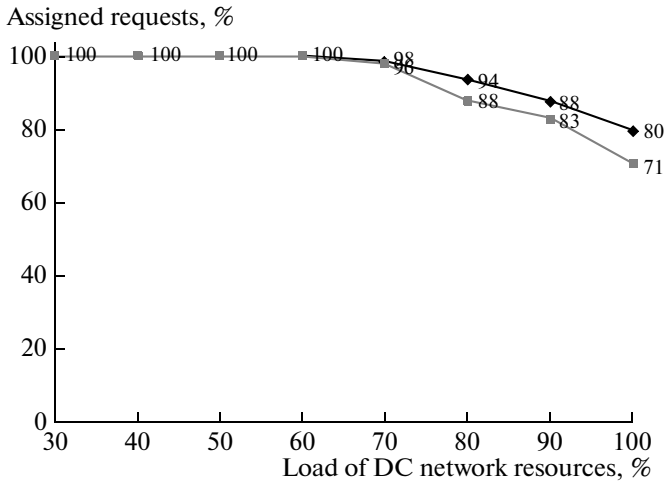

Fig. 4. Percentage of assigned requests depending on the replication procedure: –◆– algorithm with replication and –■– algorithm without replication.

To demonstrate the difference between these assignment methods, we use the DC consisting only of computation nodes with the following characteristics:

(1) the number of computation nodes is 1000;
(2) the number of requests is 100 (100 virtual machines for each request);
(3) the number of characteristics is one: ⟨CPU frequency⟩;
(4) the overall load is 50%.

For the compact assignment, the following results were obtained:

(1) virtual machines are assigned only to 500 computation nodes out of 1000 nodes;
(2) in each node to which virtual machines are assigned, the workload is close to 100%.

For the balanced assignment, the following results were obtained:

(1) virtual machines are assigned to all computation nodes;
(2) in each node, the average workload is 50%.

This example shows that the compact load makes it possible to leave some computation nodes unloaded, resulting in energy saving. The balanced load provides the decrease in the average load of nodes.

We propose to use the compact assignment when physical resources of a network are not critical resources, i.e., when the ratio of the costs of the required network resources to the available network resources is below the given threshold. Otherwise, the balanced assignment is recommended. This feature is illustrated by the experimental investigation using the DC with the following characteristics:

the FatTree architecture [2] with three levels of commutation elements;

60 computation nodes and 60 data storages;
the assignment was performed for 100 requests (on the average, 10 virtual links for each request);
the average load of the computation nodes and data storages was fixed to be 75%;
the average load of the DC network resources is varied from 30 to 100%.

The results for the compact and balanced assignment depending on the workload of virtual links are given in Fig. 3. It can be seen that, for the load of 60% and higher, the balanced assignment makes it possible to increase the percentage of assigned requests.

4.4. The Efficiency of the Replication Procedure

The efficiency of the replication procedure is investigated using the same sets of requests as in Subsection 4.3. The run was performed with the replication on and off. For each run of the algorithm, the balanced assignment was used. The results are shown in Fig. 4. It can be seen that the replication procedure provides the increase in the percentage of the assigned requests.

5. CONCLUSIONS

In this work, we extended the formulation of the problem of assigning requests to physical resources to the possibility of setting an arbitrary number of SLA requirements for each type of resources. An algorithm for assigning requests to physical resources for the DC with independent schedulers for different types of resources was proposed. Based on the exploration of the properties of the algorithm, conclusions about the domains of its efficient application are made. This algorithm can be used for designing cloud platforms, for example, in the OpenStack platform [13] or in the self-organizing cloud platform that is being developed in the Applied Research Center for Computer Networks.[4]

In order to run the algorithm and check its applicability for your data center, it is required to
(1) write a letter to ev@arccn.ru, indicating your full name;
(2) install any VNC client;
(3) read the manual.

The tool offers the possibility to describe DC resources, to create a set of requests, and to investigate how these requests will be assigned to physical resources using the chosen algorithm.

REFERENCES

1. P. M. Vdovin, I. A. Zotov, V. A. Kostenko, A. V. Plakunov, R. L. Smelyansky, "Data center resource allocation problem and approaches to its solution," in *VII Moscow Int. Conf. on Operations Research* (*ORM2013*) (Vychisl. Tsentr Ross. Akad. Nauk, Moscow, 2013), Vol. 2, pp. 30–32.
2. P. M. Vdovin, I. A. Zotov, V. A. Kostenko, A. V. Plakunov, and R. l. Smelyanskiy, "Comparing various approaches to resource allocation in data centers," J. Comput. Syst. Sci. Int. **53** (2014).
3. A. Amies, H. Sluiman, Q. G. Tong, et al., *Developing and Hosting Applications on the Cloud* (IBM Press, Boston, 2012).
4. E. Wustenhoff, "Service level agreement in the data center" (Sun BluePrints, Sun Microsystems Professional Series, 2002).
5. M. G. Furugyan, "Some algorithms for analysis and synthesis of real-time multiprocessor computing systems," Program. Comput. Software **40**, 21–27 (2014).
6. M. G. Furugyan "Some algorithms of solving minimax multiprocessor scheduling problem," J. Comput. Syst. Sci. Int. **53**, 195–200 (2014).
7. V. A. Kostenko and A. V. Plakunov, "An algorithm for constructing single machine schedules based on ant colony approach," J. Comput. Syst. Sci. Int. **52**, 928–937 (2013).
8. V. A. Kostenko, "Scheduling algorithms for real-time computing systems admitting simulation models," Program. Comput. Software **39**, No. 5, 255–273 (2013).
9. V. A. Kostenko, P. E. Shestov, "A greedy algorithm for combined scheduling of computations and data exchanges in real_time systems," J. Comput. Syst. Sci. Int. **51**, 648–662 (2012).
10. D. A. Zorin and V. A. Kostenko, "Algorithm for synthesis of real-time systems under reliability constraints," J. Comput. Syst. Sci. Int. **51**, 410-417.
11. E. G. Coffman, M. R. Garey, and D. S. Johnson, "Approximation Algorithms for Bin Packing: A Survey," in *Approximation Algorithms for NP-Hard Problems* (PWS, Boston, 1996), pp. 46–93.
12. D. Eppstein, "Finding the *k* shortest paths," SIAM J. Comput. **28**, 652–673 (2006).
13. K. Pepple, *Deploying OpenStack* (O'Reilly, Sebastopol, 2011).

Translated by Yu. Kornienko

[4] Information about the studies carried out in the Applied Research Center for Computer Networks is available on the Internet at http://arccn.ru/research/762.