NACER: a Network-Aware Cost-Efficient Resource allocation method for processing-intensive tasks in distributed clouds*

Ehsan Ahvar, Shohreh Ahvar, Noel Crespi, Joaquin Garcia-Alfaro
Institut Mines-Telecom SudParis, Evry, France

Zoltán Ádám Mann
Budapest University of Technology and Economics, Budapest, Hungary

Abstract

In the distributed cloud paradigm, data centers are geographically dispersed and interconnected over a wide-area network. Due to the geographical distribution of data centers, communication networks play an important role in distributed clouds in terms of communication cost and QoS. Large-scale, processing-intensive tasks require the cooperation of many VMs, which may be distributed in more than one data center and should communicate with each other. In this setting, the number of data centers serving the given task and the network distance among those data centers have critical impact on the communication cost, traffic and even completion time of the task. In this paper, we present the NACER algorithm, a Network-Aware Cost-Efficient Resource allocation method for optimizing the placement of large multi-VM tasks in distributed clouds. NACER builds on ideas of the A^* search algorithm from Artificial Intelligence research in order to obtain better results than typical greedy heuristics. We present extensive simulation results to compare the performance of NACER with competing heuristics and show its effectiveness.

1 Introduction

Cloud computing is an Internet-based computing paradigm where computing resources are offered over the Internet as scalable on-demand services [1].

Traditional clouds were designed in a centralized manner. They include one huge data center (DC), or a small number of huge DCs. However, we can recently witness a revolution in the design of new cloud systems. Modern cloud infrastructures are increasingly geographically distributed [2]. As smaller companies are also entering the cloud market, also the size of typical DCs becomes relatively smaller than the traditional huge DCs of cloud giants. The reasons for adopting distributed smaller-sized DCs instead of centralized mega-DCs include also lower upfront investment, shorter time-to-market, decreased vulnerability to natural disasters, and proximity to users [3, 4].

Although distributed cloud is lucrative, it is associated with several new challenges. One of the key issues is inter-DC communication, which may impact delay, cost and security. In particular, it is possible that the virtual machines (VMs) forming a large-scale task must be provisioned in multiple DCs [5]. Consequently, the cost (and also delay) of communication between the different DCs is far more significant than in the case of centralized cloud architectures where the cloud resources are concentrated inside a few large DCs. In addition, dependency across DCs, such as communication awareness or inter-VM communication, is rarely considered by existing studies [6].

This paper targets the challenge of optimizing inter-DC communication cost for large-scale tasks in distributed clouds. We focus on network-aware resource allocation as an effective tool to reduce communication cost in distributed clouds. By reducing inter-DC communication among the VMs of multi-VM tasks, overall cost can be decreased, while at the same time improving application performance and thus service quality.

We propose a Network-Aware Cost-Efficient Resource allocation method, called NACER, for distributed clouds. Our goal is to minimize the network communication cost of running a task in a distributed cloud by selecting the DCs to provision the VMs in such a way that the total network distance (hop count or any reasonable measure) among the selected DCs is minimized. Our underlying assumption is that cloud network cost is mainly determined by inter-DC traffic which occurs when VMs of the same task that must cooperate and communicate with each other are placed on different DCs. We devise two versions of NACER with slightly different objective functions: NACER1 optimizes the total communication cost among the selected DCs, whereas NACER2 weights

*Published in the Proceedings of the 14th IEEE International Symposium on Network Computing and Applications, pp. 90-97, 2015.

inter-DC communication cost with the number of affected VM pairs. NACER1 is thus more suitable for delay-critical applications, because the total network distance of the selected DCs will largely determine system latency, whereas NACER2 is more appropriate for throughput-critical applications, as the number of VM pairs requiring inter-DC communication influences system bandwidth.

Most previous work use simple greedy algorithms for optimizing VM allocation [7]. NACER is more powerful in several respects. First, it uses a cost function that balances two – sometimes conflicting – goals: minimizing the number of selected DCs by preferring DCs with high available capacity and minimizing the network distance among selected DCs by preferring DCs near the already selected ones. Second, when selecting the next DC, NACER does not only consider the communication cost between the already selected DCs and the new DC, but it also *predicts* communication cost with the DCs that will be selected later on, thus being able to make more informed decisions. For this purpose, NACER is based on the A^* search algorithm. Third, since the starting DC has large impact on the set of selected DCs (because of the bias to select DCs that are near to the ones already selected), our algorithm tries each DC as possible starting DC, and then selects the best set of DCs obtained.

We validate the approach via simulation, using CloudSim [8]. We compare the obtained results with previous approaches, and show that NACER1 and NACER2 both significantly reduce network cost while satisfying QoS constraints.

The rest of the paper is organized as follows. Section 2 describes related work, followed by the problem formulation, proof of problem hardness and related discussion in Section 3. Section 4 presents the main contribution of this paper, which is the NACER resource allocation algorithm, presented first in the NACER1 version. NACER2 is described in Section 5. The assessment of NACER1 and NACER2 is covered in Section 6, and Section 7 concludes the paper.

2 Related Work

In this section, we introduce recent attempts to design network-aware resource allocation algorithms for improving some factors such as cost, energy consumption, or completion time.

Most of the existing work on network-aware resource allocation focus on resources and communication within a single DC. In [9], the authors proposed a network-aware resource allocation algorithm with the aim of minimizing communication cost. They worked on VM placement optimization to reduce bandwidth usage. Meng et al. [10] proposed a network-aware VM placement scheme. Their goal is to reduce total network traffic by optimal VM placement. They allocate VMs with large communication near each other to reduce network traffic and cost. Dong et al. [11] proposed a network-aware VM placement scheme in consideration of multiple resource constraints inside a DC. They considered two types of resources at the same time for VM placement: the allocation of physical server resources (CPU, memory, storage etc.), and the optimization of network resources. The main goal is to improve energy consumption.

The limitation of all these work is that they do not consider inter-DC communication and the impact of geographical location. There are very few work that address communication within distributed clouds [12].

Alicherry and Lakshman [5] proposed algorithms for network-aware selection of DCs and allocation of VMs in a distributed cloud system. They focused on improving completion time of a task by reducing maximum distance between the selected DCs. The underlying assumption is that VMs which are tardy in their completion times, due to communication latencies, can increase overall completion times for user requests. In their work, they have developed DC selection algorithms for VM placement that minimizes the maximum distance between the selected DCs. Although the *maximum distance* between selected DCs may be an appropriate metric for improving completion time of some applications, we believe that for reducing overall communication cost, the *total distance* among selected DCs is a more appropriate metric. Moreover, Alicherry and Lakshman divide the problem of allocating VMs to DCs into two subproblems, namely DC selection and request partitioning, thereby potentially losing the ability to find a global optimum. They present a simple greedy heuristic for each subproblem, from which the first one is shown to be a 2-approximation algorithm, but only for the case of Euclidean distances (which applies to geographical distances, but not necessarily to network distances).

A somewhat similar approach was suggested by Aldhalaan and Menascé [13]. In this case, the aim is not to decrease cost but to increase revenues. The underlying assumption is that the user will pay higher fees if the VMs of a request are allocated near to each other. The authors propose some proprietary heuristics for this problem. Their model is hierarchical in the sense that not only DCs are selected, but also clusters, racks, and individual servers. This detailed modeling of intra-DC allocation may limit the scalability of the algorithms; indeed, they were tested on a setup with only two DCs. In contrast, our more coarse-grained model allows us to consider distributed cloud systems of much higher scales, with tens to hundreds of DCs.

3 Problem formulation

NACER is designed to support user requests with high number of VMs that communicate with each other. A typical field of our application is extreme-scale high performance scientific computing with high internal communications where the required data for a process node is available or produced in another process node. A practical example is weather, climatology and solid Earth sciences (WCES) that inspired by [14] as following. Steerable weather radars can produce data at a rate of nearly 200 Mbs. The data are sent to the backend application in the cloud, which aggregates the data feeds from multiple weather radars, processes the data, and then sends the result for steering the weather radars for subsequent monitoring. Because these high-bandwidth sensors collect data in the physical world, their produced data are directly related to unpredictable real-world events. For example, they produce more data – and thus require more VMs for processing – during thunderstorms than during calm weather. Therefore, an on-demand resource allocation mechanism for this type of application is necessary, for which distributed cloud can be ideal.

The system model under consideration is a large-scale distributed cloud computing system including N DCs with different capacities, numbered from 1 to N. These DCs are connected to each other through a communication network (e.g., VPN, leased line). The system receives requests to provision large-scale user tasks. The number of VMs to provision for the next task is denoted by M. Each DC has a current capacity, which is the number of further VMs that it can accommodate. The current capacity of DC i is denoted by S_i ($i = 1, \ldots, N$). For each pair of DCs, the cost of communication between them (e.g., the number of hops on the routing path between them) is given: $Cost_{ij}$ denotes the cost of communication between DCs i and j. In addition, a DC information table (DIT) containing information about the DCs can be available in the cloud. DIT can include information such as number of available VMs of each DC, location of DCs or distance between each pair of them. Each DC can register its information in the DIT and keep it updated.

Our aim is to select a subset $X \subseteq \{1, \ldots, N\}$ of the DCs to accommodate the newly requested VMs. The selected DCs must have sufficient total capacity to accommodate M VMs:

$$\sum_{i \in X} S_i \geq M. \tag{1}$$

The total inter-DC communication cost corresponding to selection X is given by

$$TC(X) = \sum_{i \in X} \sum_{\substack{j \in X \\ j \neq i}} Cost_{ij}. \tag{2}$$

Hence, the *Network-Aware Resource-Allocation Problem* (NARAP) consists of selecting X in such a way that constraint (1) is satisfied and the total cost given in (2) is minimized.

Theorem 1. *The NARAP problem is NP-hard in the strong sense.*

Proof. We prove the theorem by reduction from the decision version of the stable set problem, which is known to be NP-hard in the strong sense. In that problem, the input is a graph G and a number k and the question is whether there is a stable set of size k in G.

From this instance, we construct a NARAP instance as follows. Each vertex of G is represented by a DC. If two vertices are adjacent in G, then the communication cost between the corresponding DCs is 1, otherwise it is 0. The capacity of each DC is 1. Further, let $M = k$.

If there is a stable set of size k in G, then the corresponding DCs build a solution of NARAP with cost 0. Similarly, if NARAP admits a solution X with cost 0, then the vertices of G corresponding to the DCs in X form a stable set of size k in G. Hence, the stable set problem is equivalent to whether NARAP admits a solution with cost 0. □

3.1 Discussion

If possible, the allocation of all the VMs of a task into the same DC is the best solution to reduce inter-DC traffic and communication cost. However, in a distributed cloud, the DCs are relatively smaller than centeralized cloud DCs and may already have high load, so it is possible that a large-size task needs resources from multiple DCs [5]. In this situation, finding an optimal allocation taking into account the inter-DC communication cost is highly non-trivial.

Most existing approaches to VM allocation use greedy heuristics [7]. In the following, we argue that such methods are not appropriate for the NARAP problem.

One natural greedy approach is to minimize the Number of Selected DCs (NSD). This is a logical idea since fewer selected DCs will likely lead to fewer communicating VM pairs being allocated to different DCs. Minimizing NSD is fairly easy, because it involves simply selecting the DCs with the highest available capacity. However, the problem with optimizing NSD is that the selected DCs may be very far from each other (e.g., in terms of hop count), so that the resulting solution may be much worse than the optimum.

Another extreme would be to only focus on the DC-to-DC Distance (DDD), where the distance between two DCs means the number of intermediate routers, intermediate network elements, hop count or any reasonable measure between the DCs. Selecting DCs with small distance between them can eliminate the above problem. Minimizing the DDD metric also lends itself nicely to drive a greedy algorithm, in which DCs are selected one after the other, always selecting the next one based on its distance from the already selected ones. However, the problem of this way is that a large number of DCs may be selected so that the overall communication cost can again be much worse than the optimum.

As can be seen, the challenge is to somehow combine these two, sometimes conflicting, objectives: we should select a relatively small number of high-capacity DCs but at the same time also make sure that their DDD is small.

Moreover, greedy algorithms always suffer from the weakness that they make only local decisions and therefore they usually do not find a global optimum. In our case, selecting DCs one by one, even if the next selected DC is based on optimizing a combination of the NSD and DDD metrics, will likely miss global optima if only the already selected DCs are taken into account in the cost calculation. This is because DCs that will be selected later on by the algorithm will also contribute to the overall cost, but such future costs are not taken into account by typical greedy methods. Hence, we need a more intelligent approach that also estimates the cost incurred by later DC selection steps.

In the next section, we show how we managed to address these challenges with an intelligent algorithm.

4 Network-Aware Cost-Efficient Resource allocation (NACER1)

In this section, we present our Network-Aware Cost-Efficient Resource Allocation method, in its first version (NACER1) that is designed to select a cost-efficient subset of DCs for placing VMs of a large-scale processing- and communication-intensive task. To describe NACER1, first, we introduce its general three-step mechanism. Then we present the NACER1 logical architecture and describe each of its components in detail.

4.1 General mechanism

This section shows a high-level overview of NACER1. NACER1 runs three main steps one-by-one to allocate the required M VMs to appropriate DCs:

- *Step.1*—Creating a graph representation of the distributed cloud.
- *Step.2*—Finding candidate subgraphs.
- *Step.3*—Selecting the best subgraph.

Next, we describe each step.

4.1.1 Creating a graph representation of the distributed cloud

This step maps cloud DCs to a vertex- and edge-weighted complete graph $G = (V, E, S, Cost)$. The vertices of G correspond to the DCs: $V = \{v_1, \ldots, v_N\}$, where v_i represents the ith DC. Each vertex v_i is labeled with the available capacity of the given DC ($S(v_i)$). Each pair of vertices v_i, v_j is connected by an edge e_{ij}, labeled with the communication cost between the given DCs ($Cost(e_{ij})$).

4.1.2 Finding candidate subgraphs

The aim of this step is to find, from each vertex v as starting point, an induced subgraph $G'(v)$ of G. The vertices of $G'(v)$ must have sufficient total capacity to accommodate the M VMs; furthermore, the total communication cost of the edges of $G'(v)$ should be as low as possible. This is where the challenges described in Section 3.1 are addressed: we use an intelligent heuristic borrowing ideas from Artificial Intelligence research, specifically the A^* search algorithm [15], described in more details in Section 4.3. Our algorithm is more powerful than simple greedy methods focusing only on NSD or DDD, and it also predicts cost of its future allocation steps, thus being able to overcome local optima.

4.1.3 Selecting the best subgraph

NACER1 compares the total communication cost of the candidate subgraphs found in Step.2, and selects the subgraph whose total communication cost is minimum. This is important because the candidate subgraph formed starting from v_i will often be biased towards DCs in the proximity of DC i; taking the best one of the candidate subsets helps to find a globally optimal subset.

In principle, it would also be possible to consider all subgraphs of G with sufficient total capacity and choose the one with lowest total communication cost. However, the number of all such subgraphs can be exponential, making this approach intractable in practice. In contrast, our method is a fast, polynomial-time heuristic.

4.2 NACER architecture

Our proposed NACER method consists of the following components:

- *(1) a DC Graph Creator (DGC),*
- *(2) a Candidate Subgraph Creator (CSC),*
- *(3) a Prediction-based DC Selector (PDS),*
- *(4) a Best Subgraph Selector (BSS) and*
- *(5) a DC Information Table (DIT).*

All information coming from the cloud manager is stored in the DIT. Based on information stored in the DIT, the DGC module creates the graph representation. After that, the CSC module finds candidate subgraphs. To create a subgraph, CSC repeatedly calls the PDS module, which is responsible for extending the current the subgraph by one more DC. Finally, the BSS module selects the best candidate subgraph in terms of communication cost.

Next, we describe the modules in more detail.

4.3 Candidate Subgraph Creator (CSC)

The aim of CSC is to determine for each vertex v as starting point an induced subgraph $G'(v)$ of G with sufficient total capacity and low communication cost. $G'(v)$ is grown from $\{v\}$ iteratively by adding one vertex a time. The already selected vertices are stored in a set Al; initially, $Al = \{v\}$. In each step, CSC checks whether the vertices already selected have sufficient total capacity. If yes, the creation of $G'(v)$ is finished. Otherwise, the PDS module is called to select one more vertex for inclusion in Al, and the cycle continues, until the total capacity of the selected DCs is sufficient. Then, $G'(v)$ is the subgraph induced by Al.

This way, a candidate subgraph is created for each vertex as starting points, yielding altogether N candidate subgraphs (see Algorithm 1).

Algorithm 1: CSC Module Mechanism

Input : Vertex- and edge-weighted complete graph $G = (V, E, S, Cost)$ with $|V| = N$
Input : Number of VMs to allocate (M)
Output: N candidate subgraphs with capacity at least M

```
foreach v ∈ V do
    Let Al = {v};
    Let TS = S(v);
    while TS < M do
        Let w = PDS(Al); /*selects one more DC*/
        add w to Al;
        Let TS = TS + S(w);
    Let G'(v) = G[Al]; /*induced subgraph*/
return candidate subgraphs {G'(v) : v ∈ V}
```

4.4 Prediction-based DC Selector (PDS) module

Whenever CSC needs to add a new DC to the candidate subgraph being generated, it calls the PDS module. PDS aims at selecting the most cost-effective DC to be included in the subgraph.

Let G' denote the current status of the subgraph being created by CSC, let Al denote the set of vertices in G' (i.e., the set of already selected DCs) and $P = V \setminus Al$ denotes the set of DCs that are still available for selection.

Further, let $z = |Al|$. PDS is invoked if the total capacity of these z DCs is not sufficient to accommodate all the M VMs; the aim is to select the next DC from the ones in P.

The PDS mechanism is based on the A^* algorithm [15]. It uses an evaluation function for the possible choices that combines the costs already incurred and an estimate of the costs that will be incurred in the future. The algorithm takes the choice with the smallest value of the evaluation function.

For each possible choice $v \in P$, the evaluation function is

$$c(v) = g(v) + h(v). \tag{3}$$

Here, $g(v)$ is the communication cost between the already selected DCs and v. This cost will be certainly incurred if v is selected. $g(v)$ can be easily calculated as follows:

$$g(v) = \sum_{w \in Al} Cost(vw). \tag{4}$$

The function $h(v)$ is an estimate of the communication cost caused by the further DCs that we will have to select later on to accommodate all the M VMs. This is estimated as

$$h(v) = NE \cdot ED, \tag{5}$$

where NE is the estimated number of edges that will be added later to the subgraph, in the course of allocating the remaining VMs, and ED is the estimated average communication cost for these new edges. To estimate NE, recall that G is a complete graph, so that each new node added to a subgraph with k vertices will add k new edges. After adding v to the subgraph, it will consist of $z+1$ vertices, so adding further vertices will lead to $z+1, z+2, \ldots$ new edges. Hence, if y further DCs will have to be selected after v, we have

$$NE = \sum_{k=z+1}^{(z+1)+(y-1)} k = z \cdot y + \frac{y \cdot (y+1)}{2}. \tag{6}$$

Here, y can be estimated as follows:

$$y = \frac{M - ((\sum_{w \in Al} S(w)) + S(v))}{AvgS}, \tag{7}$$

where M is the total number of VMs needed for the user request, $\sum_{w \in Al} S(w)$ is the number of allocated VMs till now for the user request, $S(v)$ is the number of VMs that can be allocated if DC v is chosen next, and $AvgS$ is the average capacity of all DCs.

It remains to estimate ED, the average communication cost of the edges that will be added to the subgraph in subsequent steps. One possibility is to use the average communication cost among all DCs. This would be a good estimate if we sampled edges randomly. However, our algorithm is biased towards edges of lower cost, so that the overall average may be an overestimate. We can get a more accurate estimation by calculating the average cost of the edges that the algorithm has selected so far, i.e., the edges within a set Al' that includes the subgraph G' elements as well as the candidate v. However, when selecting the second DC, G' has only one vertex and no edge, so in this case, we use the average communication cost between the first DC and all other DCs.

$$ED = \begin{cases} \dfrac{\sum\limits_{v_i \in Al'} \sum\limits_{\substack{v_j \in Al' \\ v_j \neq v_i}} Cost(v_i v_j)}{z(z+1)/2} & \text{if } z > 1 \\ \dfrac{\sum\limits_{w \in P} Cost(sw)}{N-1} & \text{if } z = 1, Al' = \{s\} \end{cases} \tag{8}$$

Putting all the pieces together, we get a fairly good estimate of the total cost of selecting DC v next. Based on these estimates, the algorithm can select the best choice (see Algorithm 2).

4.5 Best Sub-graph Selector (BSS) module

This module computes the total communication cost of all created subgraphs and selects the one with the lowest cost.

Algorithm 2: PDS Module Mechanism

Input : Complete graph $G = (V, E, S, Cost)$
Input : Set Al of z already selected DCs
Output: Selected DC

$cost_{min} = \infty$;
foreach $v \in V \setminus Al$ **do**
    compute $c(v)$ using Equations (3)-(8);
    if $c(v) < cost_{min}$ **then**
        $cost_{min} = c(v)$;
        selected $= v$;
return selected

4.6 Datacenters Information Table (DIT)

DIT holds all information of the cloud to be used by NACER1.

5 NACER2

NACER1 is designed to work without using detailed communication intensity information on the level of individual VM pairs (often referred to as traffic matrix). This is because recent studies [16] [9] show that assuming that the cloud provider knows VM traffic is a very strong assumption and typically not realistic. NACER1 aims at placing VMs of a task as near as possible to each other and this practical rule generally leads to an improvement in communication cost. Of course, if detailed VM traffic information is available, then more informed placement

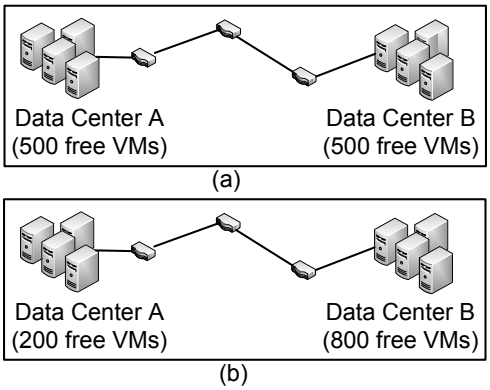


Figure 1: Communication cost considering inter-VM relations

decisions can be made – although this comes with a dramatic increase in the size of the search space. Nevertheless, for certain applications, a good compromise can be found by *weighting* the cost of communication between a pair of DCs with the *number of affected VM pairs*.

As an example, consider the two cases in Figure 1. In both cases, we place altogether 1,000 VMs in the two DCs. The network distance (e.g., hop count) between the two DCs is the same in both cases. However, in case (a), the number of VM pairs affected by this network distance is $500 \cdot 500 = 250,000$; in case (b), the number of affected VM pairs is only $200 \cdot 800 = 160,000$. Accordingly, the overall communication cost will be lower in case (b) than in case (a).

This is the basic idea behind NACER2. We assume that a base communication cost $Cost_0(DC_i, DC_j)$ is given for each pair of DCs (DC_i, DC_j). The real communication cost is weighted by the number of VM pairs that are placed on the DCs:

$$Cost(DC_i, DC_j) = Cost_0(DC_i, DC_j) \cdot K(DC_i) \cdot K(DC_j),$$

where $K(DC_i)$ denotes the number of VMs placed on the given DC. Beyond this change to the cost calculation, NACER2 works in the same way as NACER1.

We believe that NACER2 is a good compromise between accuracy and efficiency. We still do not need detailed VM-level traffic information, which would be infeasible to obtain and would blow up the search space. NACER2 relies on information that is readily available and does not make decisions on the level of single VMs. On the other hand, NACER2 can make more accurate decisions than NACER1 in the case of applications where the number of VM pairs that are placed in different DCs gives a good estimate of the total inter-DC communication cost.

This is the case for applications in which there is intensive communication among many VM pairs and the VMs behave mostly in a homogeneous way. Examples of such applications include large-scale scientific experiments and engineering simulation.

6 Performance Evaluation

In this section, we evaluate the performance of NACER1 and NACER2 by comparing against two other resource allocation algorithms: Random and Greedy.

The Random resource allocation algorithm starts by selecting a DC randomly and placing as many VMs as possible in the selected DC. If not all VMs could be allocated in the selected DC, then a further DC is selected, again randomly, to place the remaining VMs. This process is repeated until the requested number of VMs is placed.

The Greedy algorithm selects the DC with maximum free capacity and allocates as many VMs from the request as possible in the selected DC. If further VMs are necessary, then the Greedy algorithm selects from the remaining DCs again the one with maximum free capacity. This process continues until all the VMs are placed [5]. The Greedy algorithm thus finds the optimal allocation according to the NSD metric.

6.1 Simulation Model

For our experiments, we used the CloudSim simulator [8]. We consider four different configurations of distributed cloud, consisting of 25, 50, 75, and 100 DCs, respectively. DCs are connected thought leased lines. Three different request sizes (tasks) with 1,000, 1,500, and 2,000 VMs are considered. The capacity of the whole distributed cloud is chosen randomly, between 5,000 and 30,000, in each run. This total capacity is divided among the DCs: for a cloud containing 100 DCs, each DC has capacity between 50 and 300 VMs, for 50 DCs, each DC has capacity from 100 to 600 VMs, for 25 DCs, each DC has capacity between 200 and 1200.

It is worth highlighting that, even though the capacity of the DCs is set randomly, once set, it remains fixed across the runs of all tested algorithms, to ensure comparability of the results. We conducted five different tests as described below. For each test, we report the results as average of 100 runs on our server with 32-core Intel Xeon Processor at 2.7Gigahertz, 256GB RAM, and 2.5TB hard drive.

Test 1: Total VM-to-VM communication cost—This test is one of the main indicators of the effectiveness of the resource allocation schemes in terms of cost, network traffic, and cloud performance. Algorithms that reduce the total VM communication for all VM pairs of a task offer lower communication cost.

Test 2: Total DC-to-DC communication cost—This test is a slightly modified version of Test 1. It shows the general cost of communication between the selected DCs for processing the given request, without assuming any knowledge of the communication intensity among specific VM pairs. Algorithms that reduce the total hop count among the selected DCs offer lower communication cost.

Test 3: Number of selected DCs (NSD)—As explained earlier, reducing NSD is a means to reduce overall communication cost. Reducing NSD is helpful, but in itself not sufficient, for reduction of total communication cost. This test shows the algorithms' ability to reduce NSD.

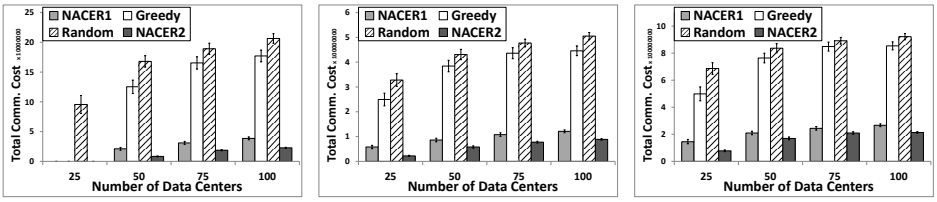
Test 4: Average network distance of DCs (DDD)—As explained earlier, reducing DDD is another means to reduce overall communication cost. Reducing DDD is helpful, but in itself not sufficient, for reduction of total communication cost. This test shows the algorithms' ability to reduce DDD.

Test 5: Maximum network distance between DCs—Even if the average network distance between the selected DCs is low, it is possible that there is a pair of DCs with high network distance. This can have strong negative impact on the communication between some pairs of VMs, making these VMs lag behind others. For applications where the makespan is critical, i.e., application performance is determined by the VM that finishes last, the maximum of the network distances between DCs is hence more important than the average.

6.2 Simulation Results

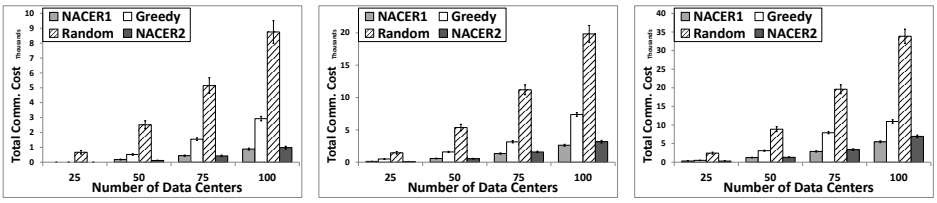
Figures 2 and 3 show the results of Test 1 (total VM-to-VM communication cost) and Test 2 (total DC-to-DC communication cost), respectively. It can be clearly seen that in both cases, Random performs worst, Greedy consistently outperforms Random, and the two versions of NACER perform best, with sizeable improvement over both Random and Greedy. NACER2 delivers better results than NACER1 concerning total VM-to-VM communication cost; on the other hand, concerning total DC-to-DC communication cost, NACER1 is better than NACER2. This is not surprising, since NACER1 explicitly aims at optimizing total DC-to-DC communication cost, whereas the objective function of NACER2 is weighted with the number of affected VM pairs.

Figures 4 and 5 help us to understand the reasons for the superior performance of NACER over Random and Greedy. In particular, Figure 4 shows the results of Test 3 (NSD metric). As expected, Greedy performs best



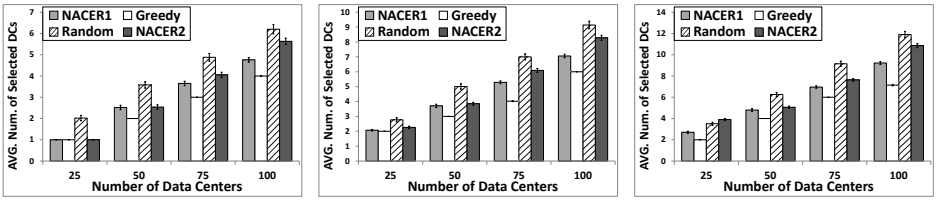
(a) $M = 1000$ (b) $M = 1500$ (c) $M = 2000$

Figure 2: Test 1—Total VM-to-VM communication cost for a task with M VMs



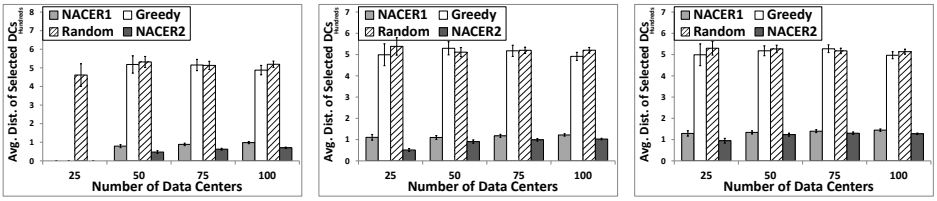
(a) $M = 1000$ (b) $M = 1500$ (c) $M = 2000$

Figure 3: Test 2—Total DC-to-DC communication cost for a task with M VMs



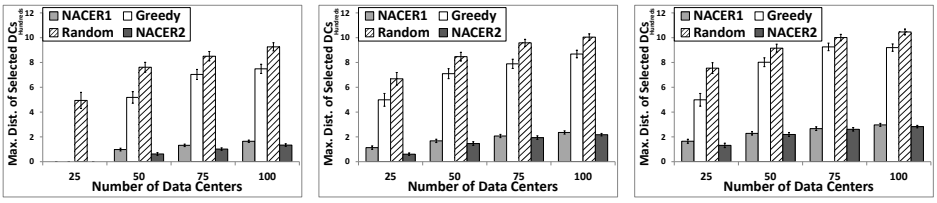
(a) $M = 1000$ (b) $M = 1500$ (c) $M = 2000$

Figure 4: Test 3—Number of selected DCs (NSD metric) for a task with M VMs



(a) $M = 1000$ (b) $M = 1500$ (c) $M = 2000$

Figure 5: Test 4—Average network distance between selected DCs (DDD metric) for a task with M VMs



(a) $M = 1000$ (b) $M = 1500$ (c) $M = 2000$

Figure 6: Test 5—Maximum network distance between selected DCs for a task with M VMs

in this respect, since it always delivers optimal results. However, it is important to observe that the advantage of Greedy over NACER is not so high. This is remarkable since NACER does not aim explicitly at optimizing NSD. Figure 5 shows the results of Test 4 (DDD metric). Here, it can be observed that NACER significantly outperforms both Random and Greedy. Since the total communication cost is determined by both NSD and DDD, and Greedy performs only slightly better than NACER on NSD but much worse on DDD, this leads to the overall superior behavior of NACER. This also justifies our initial intuition that optimizing just the NSD without taking into account DDD is not sufficient for overall good performance.

It is also interesting to look at the effect of the DCs' capacity. If there is a DC with sufficient capacity to accommodate all VMs, then Greedy, NACER1, and NACER2 all find this solution. This is the case when $M = 1000$ and we have 25 DCs with capacities ranging between 200 and 1200. It can be seen in Figure 4(a) that the number of selected DCs is indeed 1 for all of the three algorithms. Accordingly, communication cost is 0, as can be seen in Figures 2(a), 3(a), and 5(a). However, when the DCs' capacity decreases, the difference between Greedy and NACER becomes apparent: since NACER1 and NACER2 also consider DDD as an important metric in their decision, they can achieve better results than Greedy in distributed cloud environment with high number of low-capacity DCs.

In addition to communication cost, completion time of processing a task is another important factor. The maximum network distance (i.e., hop count) between the selected DCs has a direct effect on completion time as it determines the maximum communication delay. This was the motivation for Test 5, the results of which are shown in Fig. 6. As can be seen, both NACER1 and NACER2 offer significantly lower maximum DC-to-DC distance than the other allocation methods. Note that this is not an explicit optimization objective of NACER, but a positive side effect.

Last but not least, the confidence intervals – which are also shown in the figures – of both NACER1 and NACER2 in Tests 1, 2, 4, and 5 are smaller than the confidence intervals of the other algorithms. This shows that the results of NACER1 and NACER2 are more stable. In Test 3, Greedy always offers minimum number of selected DCs and hence it has the smallest confidence intervals.

7 Conclusion

In this paper, we have addressed the problem of allocating virtual machines (VMs) in distributed clouds with the aim of minimizing inter-DC communication cost. We have shown that combining multiple metrics, namely NSD and DDD, can improve overall communication cost and even completion time of large-scale tasks. We formalized the problem and proved its NP-hardness. Getting motivation from the A^* algorithm, we have proposed a network-aware and cost-efficient resource allocation method, called NACER, to reduce communication cost of large tasks. NACER is more sophisticated than typical greedy heuristics for VM allocation, because it also predicts the communication cost that will be incurred by future DC selection decisions. Simulation results reinforced that the proposed method can considerably reduce communication cost in comparison to other allocation methods.

In general, the cost is considered as one of the major challenges for the cloud providers. However, cloud network plays a big role for geographically distributed clouds in terms of cost, performance and QoS but, to have a more accurate cost efficient system, the cost caused by computing resources should be considered as well. For our future work, we plan to extend our current cost model to a more comprehensive cost model including both communication and computing cost in distributed cloud and, then, offer cost-efficient VM allocation algorithms based on our comprehensive model.

Acknowledgment

The work of Zoltán Ádám Mann was partially supported by the Hungarian Scientific Research Fund (Grant Nr. OTKA 108947). We acknowledge as well support from the Spanish Ministry of Science (project TIN2011-27076-C03-02 COPRIVACY).

References

[1] Shangguang Wang, Zibin Zheng, Qibo Sun, Hua Zou, and Fangchun Yang. Cloud model for service selection. In *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 666–671, 2011.

[2] Iyswarya Narayanan, Aman Kansal, Anand Sivasubramaniam, Bhuvan Urgaonkar, and Sriram Govindan. Towards a leaner geo-distributed cloud infrastructure. In *Proceedings of the Sixth USENIX Conference on Hot Topics in Cloud Computing (HOTCLOUD 2014)*, 2014.

[3] Kuan-yin Chen, Yang Xu, Kang Xi, and H. Jonathan Chao. Intelligent virtual machine placement for cost efficiency in geo-distributed cloud systems. In *IEEE International Conference on Communications (ICC)*, pages 3498–3503, 2013.

[4] Kenneth Church, Albert Greenberg, and James Hamilton. On delivering embarrassingly distributed cloud services. In *Seventh ACM Workshop on Hot Topics in Networks (HotNets-VII)*, pages 55–60, 2008.

[5] Mansoor Alicherry and T. V. Lakshman. Network aware resource allocation in distributed clouds. In *IEEE INFOCOM*, pages 963–971, 2012.

[6] Jianhai Chen, Kevin Chiew, Deshi Ye, Liangwei Zhu, and Wenzhi Chen. AAGA: Affinity-aware grouping for allocation of virtual machines. In *IEEE 27th International Conference on Advanced Information Networking and Applications*, pages 235–242, 2013.

[7] Zoltán Ádám Mann. Allocation of virtual machines in cloud data centers – a survey of problem models and optimization algorithms. *ACM Computing Surveys*, 48(1), 2015.

[8] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[9] Kun You, Bin Tang, and Feng Ding. Near-optimal virtual machine placement with product traffic pattern in data centers. In *IEEE International Conference on Communications (ICC)*, pages 3705–3709, 2013.

[10] Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *IEEE INFOCOM*, pages 1–9, 2010.

[11] Jiankang Dong, Xing Jin, Hongbo Wang, Yangyang Li, Peng Zhang, and Shiduan Cheng. Energy-saving virtual machine placement in cloud data centers. In *13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 618–624, 2013.

[12] Zoltán Ádám Mann. Modeling the virtual machine allocation problem. In *Proceedings of the International Conference on Mathematical Methods, Mathematical Models and Simulation in Science and Engineering*, pages 102–106, 2015.

[13] Arwa Aldhalaan and Daniel A. Menasce. Autonomic allocation of communicating virtual machines in hierarchical cloud data centers. In *IEEE International Conference on Cloud and Autonomic Computing (ICCAC)*, pages 161–171, 2014.

[14] David Irwin, Prashant Shenoy, Emmanuel Cecchet, and Michael Zink. Resource management in data-intensive clouds: Opportunities and challanges. In *17th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN)*, 2010.

[15] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2010.

[16] Xin Li, Jie Wu, Shaojie Tang, and Sanglu Lu. Let's stay together: Towards traffic aware virtual machine placement in data centers. In *IEEE INFOCOM*, pages 1842–1850, 2014.