electronics

MDPI

Article

Swarm-Like Distributed Algorithm for Scheduling a Microservice-Based Application to the Cloud Servers

Marian Rusek * and Grzegorz Dwornicki

Institute of Information Technology, Warsaw University of Life Sciences—SGGW, ul. Nowoursynowska 159, 02-776 Warsaw, Poland; gd1100@gmail.com

* Correspondence: marian_rusek@sggw.edu.pl; Tel.: +48-22-593-7294

Abstract: Introduction of virtualization containers and container orchestrators fundamentally changed the landscape of cloud application development. Containers provide an ideal way for practical implementation of microservice-based architecture, which allows for repeatable, generic patterns that make the development of reliable, distributed applications more approachable and efficient. Orchestrators allow for shifting the accidental complexity from inside of an application into the automated cloud infrastructure. Existing container orchestrators are centralized systems that schedule containers to the cloud servers only at their startup. In this paper, we propose a swarm-like distributed cloud management system that uses live migration of containers to dynamically reassign application components to the different servers. It is based on the idea of "pheromone" robots. An additional mobile agent process is placed inside each application container to control the migration process. The number of parallel container migrations needed to reach an optimal state of the cloud is obtained using models, experiments, and simulations. We show that in the most common scenarios the proposed swarm-like algorithm performs better than existing systems, and due to its architecture it is also more scalable and resilient to container death. It also adapts to the influx of containers and addition of new servers to the cloud automatically.

Keywords: swarm intelligence; mobile agents; cloud orchestrators

Citation: Rusek, M.; Dwornicki, G. Swarm-Like Distributed Algorithm for Scheduling a Microservice-Based Application to the Cloud Servers. *Electronics* **2021**, *10*, 1553. https://doi.org/10.3390/electronics10131553

Academic Editor: Filipe Araujo

Received: 27 May 2021
Accepted: 24 June 2021
Published: 27 June 2021

1. Introduction

The cloud can be viewed as an easily usable and accessible pool of resources that can be configured dynamically. The more work needs to be done, the more resources can be acquired. However, exposing hardware as a service to other users, especially to untrusted ones, may not be a good idea because of security risks involved. Infrastructure as a service (IaaS) cloud computing service model allows the user to obtain access to virtual machines instances and install his own operating systems inside them without having the security risks of the hardware as a service (HaaS) cloud [1].

Public cloud virtual machines startup takes from tens of seconds to several minutes, because they boot a full operating system every time. Recently the cloud industry is moving beyond self-contained, isolated, and monolithic virtual machine images in favor of container-type virtualization [2]. Virtualization containers do not contain any operating system kernel, which makes them faster and more agile than virtual machines. The container startup time is about a second. Containers keep applications and their runtime components together by combining lightweight application isolation with an image-based deployment method. The most popular container platform is Docker.

The role of orchestration and scheduling within the popular container platforms is to match applications to resources. One of the most in-demand container orchestration tools available today is Google Kubernetes [3]. At a very high level, Kubernetes has the following main components: one or more master nodes, one or more worker nodes, and a distributed key-value etcd data store. The role of the Kubernetes scheduler is to assign new containers

to cloud nodes. The scheduler running on a master node obtains resource usage data for each worker node in the cluster from the etcd data store. The scheduler also receives the new container's requirements from the API server. After successful node choice and a new container startup, the resulting state of the Kubernetes cluster is saved in the etcd data store for persistence. Kubernetes is extensible and its micro-service architecture allows the use of a custom container scheduler.

Cloud platform node roles increase the deployment complexity. Thus, there exist first interesting ideas that elastic container platforms should be more like Peer-to-Peer (P2P) networks composed of equally privileged and equipotent nodes. Some authors believe that this could be a fruitful research direction [4]. A resurgence of interest in decentralized approaches known from P2P related research is clearly observable. This seems to change and might be an indicator where cloud computing could be heading in the future [5]. In this paper, such a P2P-like decentralized system for scheduling of containerized microservices is proposed. There is no distinction between master and worker nodes: all nodes are equal. There is no database, where the cluster state is saved. There is no central scheduling system and hence the containers can dynamically (re)schedule themselves during their lifetime.

Most existing container orchestrators schedule containers to run on hosts in the cluster based on resources availability. For example Docker's native clustering solution Docker Swarm [6] ships with two main scheduling strategies, spread and bin pack. The spread strategy will attempt to distribute containers evenly across hosts, whereas the bin pack strategy will place containers on the most-loaded host that still has enough resources to run the given containers. The advantage of spread is that should a host go down, the number of affected containers is minimized. The advantages of bin pack are that it drives resource usage up and can help ensure there is spare capacity for running containers that require significant resources. In this paper, we focus on the bin pack strategy of container scheduling.

The commercial schedulers mentioned above are interesting because of their maturity. Numerous clients from small to big infrastructures use them in production. However, they are also limited in terms of features. For example, none of them proposes any dynamic scheduling feature. This means that the scheduler cannot revise the current placement when the environment is changing (typically when a load spike occurs or when the client expectations change). The containers can only be stopped and started again on new nodes. In this way their runtime (i.e., volatile) state is lost. The scheduler proposed in this paper allows for such state preserving dynamic reconfiguration using container live migration.

One of the original motivation behind the use of live migration to perform dynamic scheduling of virtual machines was green computing [7,8]. For example in [9] adaptive heuristics for dynamic consolidation of virtual machines significantly reducing energy consumption by the cloud servers have been proposed. The efficiency of these algorithms was evaluated by extensive simulations using CloudSim framework [10,11], which were based on real-world workload traces from PlanetLab virtual machines. The distributed scheduler proposed in this paper regularly analyses the environment to check if a better container consolidation on cloud nodes is possible. Unused nodes can be switched off.

The problem the cloud schedulers address is an instance of the vector packing problem, a NP-hard problem in a strong sense [12]. Thus, the scheduler latency necessarily increases exponentially with the size of the cloud infrastructure. Several approaches were proposed to improve scheduler scalability. The Borg scheduler of Google runs multiple independent copies of a scheduler on a shared infrastructure and conciliate conflicts lazily [13]. Sparrow [14] proposes a two-level scheduling algorithm to schedule jobs at a very low latency. DVMS [15] leverages Peer-to-Peer architecture to reconfigure the virtual machine placement. However, it requires a fairly complicated deadlock detection and resolution mechanism. Firmament [16] exhibits a centralized approach that is scalable enough to support large Google-size data centers. The system proposed in this paper can be considered to be the extreme case of fully distributed schedulers. In addition to cloud application processes each container runs an additional scheduling process. Technically, it could be added

as an additional container to the pod encapsulating the application container without the need for modifying it. This is a so-called sidecar cloud design pattern [17].

Recently, an overwhelming spread of smart devices and appliances, namely Internet-of-Things (IoT), has pointed out all the limitations of cloud computing centralized paradigm [18]. The continuously generated sensor data streams need to be processed by various Artificial Intelligence (AI) algorithms of different computational complexities [19]. The idea of Edge computing is to extend cloud computing to the network edge with the aim of having the computation as close to the data sources, i.e., IoT devices, as possible [18]. Fog computing is extending Edge computing by transferring computation from the IoT devices to more powerful fixed nodes with built-in data storage, computing, and communication devices [20].

Microservices running inside virtualization containers represent a valid solution for code execution in such systems [21], and their controlled migration is expected to play an important role in future IoT deployments [22,23]. Possible migration scenarios involve moving IoT devices, such as cars and robots, in which case computation and storage service may need to be moved from one Edge node to another one in order remain in close proximity of the IoT device. Another scenario is to offload running containers from an Edge node to a Fog or Cloud node, when the Edge node is overloaded and it has a limited amount of computation power or storage capacity at runtime [19]. Resource provisioning remains the Achilles heel of efficiency for Edge and Fog computing applications [24]. Our goal in this paper is to design a self-adaptive distributed container orchestration system that could potentially be used for management of the cloud, Fog, and Edge nodes.

This paper is organized as follows. In Section 2 the distributed biology inspired swarm-like container scheduling algorithm implementing the spread strategy [25] is recalled and generalized to the case of heterogeneous containers. In Section 3 it is extended to encompass the bin pack strategy and analyzed mathematically using coupled differential equations. In Section 4 some results and conclusions from experiments using OpenVZ containers are presented. In Section 5 we study the performance and scaling of the distributed algorithm proposed in a simulated cloud. In Section 6 the discussion of the results and comparison with previous approaches is presented. We finish with some conclusions and remarks about future research directions in Section 7. Notations used in this paper are presented in Appendix A. The pseudo-code of the scheduling process running inside each container is listed in Appendix B.

2. Distributed Algorithm for a Heterogeneous Cloud

For the purpose of this paper we will come up with a very simple working definition of the cloud. It says that a cloud consists of computing nodes running virtualization containers. Let us consider a given node of the cloud and denote by $n_0^{(k)}$ its capacity, where $k = \text{CPU}, \text{RAM}, \text{HDD}, \ldots$ Let us denote by $n^{(k)}$ the number of resources of kind k used by all containers running on it. If $n^{(k)} = n_0^{(k)}$ for all k nothing happens. This situation is optimal. If $n^{(k)} > n_0^{(k)}$ for some k, then the node is overloaded and a spread-like strategy needs to be employed in order to expel excess containers to other nodes. Otherwise, if $n^{(k)} < n_0^{(k)}$ for some k, then the node is not fully used and can be potentially switched off to save electrical energy if another nodes can accommodate its containers. This is the case we will deal with in this paper.

In general, the task of allocating virtualization containers to cloud nodes can be considered to be an optimization problem. For example in [26] the authors present a multi-objective optimization model that aims to assist horizontal (changes in the number of instances) and vertical (changes in the computational resources available to instances) scaling in a cloud platform. The platform to be scaled is based on virtualization containers running inside virtual machines obtained from commercial IaaS providers. The model aims to minimize the total leasing cost of the virtual machines. The solution is found using a IBM CPLEX optimization solver.

The goal of an optimization algorithm is to minimize the value of some objective (cost) function by tuning the values of its parameters. Genetic algorithms, particle swarm optimization and self-organizing migrating algorithm (SOMA) [27] belong to the most popular global stochastic optimization algorithms. In our case, the system orchestrating containers in the cloud is distributed and no global information about the state of the nodes is known. Therefore no single-objective function can be defined. Instead the problem considered in this paper is inherently a multi-objective one. Therefore a cost function depending only on the values of $n^{(k)}$ and $n_0^{(k)}$ will be assigned to every cloud node.

In [25] we proposed to carry out load balancing in cloud container setups using a swarm model based on the idea of swarms of autonomous "pheromone" robots [28–35]. Our approach treats the containers as mobile agents moving between the cloud nodes. In addition to cloud application each container runs an additional process implementing the mobile agent intelligence. It periodically checks resource usage of the hosts on which it is running. We considered a single type of container and homogeneous nodes. Later experiments were performed for heterogeneous nodes [36], but the containers remained to be identical.

However, this swarm-like model can be easily generalized to the case of nonidentical containers. As with the Dominant Resource Fairness algorithm [37], the dominant resource kind can be found as:

$$\frac{n}{n_0} = \max_k \frac{n^{(k)}}{n_0^{(k)}} \tag{1}$$

This kind might be different for different nodes (even if their capacities are the same). Although not formulated in this way, the algorithm introduced in [25,36] can be understood as a multi-objective optimization algorithm where the cost functions are defined for each node by Equation (1). The optimization goal is to *minimize* them to a value ≤ 1 same for all nodes so they are equally loaded.

In this paper, we aim to extend this model not only to provide simple load balancing, but also define richer execution policies that will be capable of grouping the containers into as few nodes as possible. The goal is to *maximize* the least dominant resource use for each node:

$$\frac{n}{n_0} = \min_k \frac{n^{(k)}}{n_0^{(k)}} \tag{2}$$

while keeping the dominant resource use given by Equation (1) below the overload threshold. The choice of least dominant resource in the cost function allows making changes to a server packed only in one dimension (i.e., $n^{(k)} = n_0^{(k)}$ for some k), which otherwise would not accept new containers (each container consumes some resources of all kinds).

The SOMA optimization algorithm [27] is based on the self-organizing behavior of groups of individuals in a "social environment". SOMA was inspired by the competitive-cooperative behavior of intelligent creatures solving a common problem. Group of animals looking for food may be a good example. If a member of this group is more successful than the others than again all members change their trajectories towards him. This procedure is repeated until all members meet around the food source.

Individuals in the SOMA algorithm are abstract points in a N-dimensional space of optimization parameters and are initially generated at random. Each individual is evaluated by cost function and they migrate through the N-dimensional hyperplane of variables and try to find better solutions. In one strategy the leader (individual with the highest fitness) is chosen in each step of the migration loop. Then all other individuals begin to jump (with some randomness added) *towards* the leader.

In our case, containers are real entities (i.e., groups of processes) that move between the nodes of the cloud. For this purpose each container computes its migration probability. If the node is overloaded and a spread-like strategy is employed to expel excess containers to other nodes then the following expression for the migration probability can be used [25]:

$$p(n) = \frac{n - n_0}{n} \tag{3}$$

The function from Equation (3) simply maps the cost function from Equation (1) onto the interval $0 \leq p \leq 1$ (if the node is overloaded). The probability p is the same for all containers on the given node and plays the role of an repulsive "pheromone" [25]. Thus, the containers more likely move *away* from a node with high value of the objective function n/n_0 (lowering it). After container migration the value of p decreases on the initial node, but increases on the target node.

If the node is not overloaded and not fully loaded then it can be potentially switched off to save electrical energy if another nodes can accommodate its containers. To implement bin pack-like strategy we propose the migration probability that decreases linearly from 1 at $n = n_{\min}$ (single container on the host) to 0 at $n = n_0$ (node is fully loaded):

$$p(n) = \frac{n_0 - n}{n_0 - n_{\min}} \tag{4}$$

where n and n_0 are given by Equation (2). In this case the containers more likely move away from a node with low value of the objective function n/n_0 (lowering it even further). After migration the value of p increases on the initial node, but decreases on the target node. The form of the "pheromone" function given by Equation (4) will be investigated (and modified) in the following section using a simple mathematical model of the cloud.

3. Mathematical Model of a Homogeneous Cloud

To come up with a model we will assume that all nodes and containers are identical. Thus, the dominant resource from Equation (1) will be simply the number of containers. Moreover the numbers of containers on different nodes will not be treated as discrete variables, but rather as continuous functions of time. In addition, the destination node will be chosen randomly. Theses assumptions allow us to describe the dynamics of the cloud using a system of coupled differential rate equations. The time change of the occupation n_i of the i-th node reads as:

$$T \frac{d}{dt} n_i = -p(n_i)\, n_i + \frac{1}{N-1} \sum_{\substack{j=1 \\ j \neq i}}^{N} p(n_j)\, n_j, \quad i = 1, \ldots, N \tag{5}$$

The first term on the right side of Equation (5) are containers migrating away from the i-th node to other nodes. The second term describes the containers incoming randomly to the i-th node from other nodes. T denotes a characteristic time scale related to the container migration time, and N is the number of nodes. Let us now substitute into Equation (5) the definition of p from Equation (3). This spread strategy was studied extensively in our previous paper [38]. It leads to a system of linear equations and the solutions decay exponentially (negative eigenvalues) with time leading to a stable stationary state $n_i = n_0$ for all nodes (eigenvalue zero).

We will now focus on the more interesting bin pack strategy. It turns out that the proper choice of the repulsive "pheromone" function is not necessarily a trivial task in this case. Let us first consider a cluster of $N = 2$ nodes, one of which can be empty:

$$n_1 + n_2 = n_0 \tag{6}$$

The system of Equation (5) together with Equations (4) and (6) gives only a static solution, regardless of the initial state.

More interesting solutions appear for $N = 3$ nodes, one of which can be empty:

$$n_1 + n_2 + n_3 = 2\, n_0 \tag{7}$$

Equations (5) reduce to the system of two coupled Bernoulli equations, which is chaotic one and therefore extremely sensitive to the initial conditions. It has the desired stationary solutions:

$$\begin{aligned} n_i &= 0 \\ n_j &= n_0, \quad j \neq i, \quad j = 1, \ldots, N \end{aligned} \tag{8}$$

but the corresponding fixed points of Equations (5) are saddles (the Jacobi matrix has one negative eigenvalue, one positive, and one zero).

To deal with this instability we propose a different "pheromone" function implementing the bin pack strategy. Let us square the right hand side of Equation (4):

$$p(n) = \left(\frac{n_0 - n}{n_0 - n_{\min}} \right)^2 \tag{9}$$

Please note that for identical containers $n_{\min} = 1$ in Equation (9). For two nodes $N = 2$ the system of Equations (5) together with the initial condition Equation (6) now reduces to the equation:

$$T \frac{d}{dt} n_1 = \frac{(n_0 - n_1)(2\, n_1 - n_0)}{(n_0 - 1)^2}\, n_1 \tag{10}$$

which has the following solution:

$$n_1(t) = \frac{n_0}{2} \left\{ 1 \pm \left[\alpha\, e^{-\beta\, t/T} + 1 \right]^{-\frac{1}{2}} \right\} \tag{11}$$

where:

$$\alpha = \left(\frac{n_0}{2\, n_1(0) - n_0} \right)^2 - 1 \tag{12}$$

$$\beta = \left(\frac{n_0}{n_0 - 1} \right)^2 \tag{13}$$

It is seen from inspection of Equation (10) that it has three fixed points: one unstable $n_1 = n_0/2$ and two stable $n_1 = 0$ and $n_1 = n_0$. The corresponding solutions are distinguished in Equation (11) by the choice of the $\pm$ sign. For large t the function from Equation (11) corresponding to the $-$ sign decays exponentially. Therefore the bin-packing algorithm for two nodes has logarithmic time complexity. As we will see in Section 5 this holds also for a larger cloud.

For three hosts $N = 3$ the system of Equation (5) together with the "pheromone" function Equation (9) has also three stable fixed points corresponding to one empty node and the remaining full Equation (8) (one eigenvalue of the Jacobi matrix is negative and the remaining zero). There is also and unstable fixed point corresponding to the equal occupation of all nodes:

$$n_i = \frac{2}{3} n_0, \quad i = 1, \ldots, N \tag{14}$$

and three unwanted stable fixed points:

$$\begin{aligned} n_i &= \frac{4}{3} n_0 \\ n_j &= \frac{1}{3} n_0, \quad j \neq i, \quad j = 1, \ldots, N \end{aligned} \tag{15}$$

To prevent the system from reaching these unwanted final configurations of the cloud the container jumps cannot be done randomly. The notion of an attractive "pheromone" needs to be introduced. It is not possible to do this within the coupled differential equations model. Therefore in the following section some results and conclusions from experiments

using OpenVZ containers will be presented. It turns out that this modification also greatly improves the performance of the algorithm.

4. Experimental Results for a Small Cloud

Process migration is an old and well researched subject [39]. During container migration a group of processes is transferred from one server to another. Processes running inside virtualization containers are isolated from the rest of the system using special kernel mechanisms like, e.g., namespaces and control groups in the case of Linux. Therefore in theory their migration should be easier than ordinary processes. At the time the experiments described here were performed, Docker offered some support for checkpointing a running container to the disk, and restoring it to memory using CRIU (https://www.redhat.com/en/blog/checkpointrestore-container-migration, accessed on 26 June 2021). However, our experience was that this could not be done reliably.

Live migration refers to the process of moving a running virtual machine between different physical machines without disconnecting the client of an internet application running inside it. Virtualization container live migration is also possible, but is supported on some container platforms only, e.g., OpenVZ [40]. Others (like Docker) have not reached this level of maturity yet. OpenVZ system consists of a Linux kernel, user-level tools and templates, which are needed to create containers. Live migration of containers is achieved using lazy and iterative migration of memory pages.

To perform the experiments described in this section we created three identical virtualized KVM nodes and installed OpenVZ 7 on them. Next, 100 identical containers was launched on these nodes. In addition to that, on each node a custom written Python REST service was started. It allows the containers to check the occupation of each node, and ask their host node to migrate them to another node. In a large cloud these REST services will form a P2P network aiding the containers in finding a suitable destination node to migrate to. In our previous experiments with containers [25] OpenVZ 6 with a Linux kernel modified in order to add custom system functions were used.

At the DockerCon conference in 2015 a demonstration was shown during which 50,000 containers were launched on $N = 1000$ nodes (https://twitter.com/dockercon/status/666199498016821248, accessed on 26 June 2021). To replicate this case we will set $n_0 = 50$ in Equation (19). The initial state of our system of $N = 3$ nodes is:

$$\begin{aligned} n_1(0) &= 32, \\ n_2(0) &= n_3(0) = 34 \end{aligned} \tag{16}$$

The experiment ends after time t_0 if the final state is reached:

$$n_i(t_0) = n_0 \vee n_i(t_0) = 0, \quad i = 1, \ldots, N \tag{17}$$

In addition to the cloud application each container runs a scheduling process written in Python (cf., Appendix B). It periodically generates a random number r and compares it to the calculated migration probability p. If $r < p$ an migration event is triggered. Container chooses its destination node and asks the host node to migrate it there. On the destination node the initial loop is resumed. This random nature of the algorithm allows it to leave unstable equilibriums like the one from Equation (14).

Container migration is not instantaneous—it takes some time T usually of the order of seconds. Thus, the question arises how to calculate the "pheromone" value during container migration? Based on our previous experience with spread-like strategy [41] the number of containers on the i-th will be modified as follows:

$$n_i \to n_i - Q_i + R_i \tag{18}$$

where Q_i and R_i are the sizes of the emigration and immigration queues on this node (in [41] only Q_i was used). These numbers are calculated by the Python REST service running

on the node. Each container wanting to migrate puts a marker on the destination node, which allows us to calculate R_i. After migration this marker is cleared. The expressions for migration probabilities on the i-th node read as:

$$p_i = \left(\frac{n_0 - n_i}{n_0 - 1} \right)^2 \tag{19}$$

where the modified value of n_i from Equation (18) is used. Thus, the container migration as seen by the "pheromone" calculation is instantaneous. This greatly improves the performance of the system (i.e., t_0 is about 50 times less).

In our experiments the containers were about 50 MB large and a 100 Mb/s network was used to migrate them. The migration time was about 5 s and the application inside stopped working for about 1 s. If a container wants to migrate, then it chooses its destination node randomly. The results of a typical experiment are depicted in Figure 1. It is seen that not necessarily the node with the lowest initial number of containers is empty at the end. However, the time $t_0 \simeq 49\,T$ is about 50% larger than the time 32 T needed to move the 32 containers sequentially in an optimal way. Such an experiment was repeated 10,000 times yielding different plots of container numbers (only a typical example is presented here), and different values of t_0. It turns out that $\simeq 49\,T$ is the most probable value of t_0, but the distribution of stabilization times has a long tail giving an average value $\langle t_0 \rangle \simeq 150\,T$.

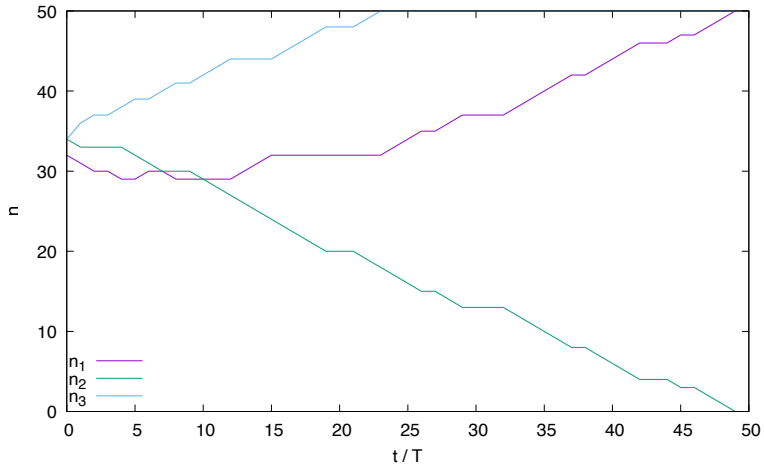

Figure 1. Numbers of containers on all nodes n versus time t.

Please note that the results depicted in Figure 1 are impossible to obtain within the mathematical model from Section 3. Indeed, for the initial state given Equation (16), the system of Equation (5) reduces to one equation similar to Equation (10) which has a solution $n_1(t) < n_2(t) = n_3(t)$ where the occupation of the first node is the lowest one. This is not the case with Figure 1 where $n_2(t_0) = 0$. However, in a real experiment some container always jumps first. Here it was the one from the second node.

We will now extend our algorithm by a more intelligent choice of the destination node by a container. This is not possible within the differential equations model (where the jumps to other nodes are random). To prevent the system from reaching the unwanted stationary state from Equation (15) migration will only be possible if there is free space on the destination node j. This means that *after* the migration:

$$n_j \leq n_0 \tag{20}$$

In our experiments with 100 containers on three nodes, this improved the average stabilization time almost two times. The best speedup is achieved, if the value of n_j given by Equation (18) is used in Equation (20). It follows from our experiments that the performance of the bin pack-like strategy is further improved (about 5 times as compared

to the random choice) if destination node j is chosen in such a way that its occupation *after* the migration is larger than the occupation of the initial host i *before* the migration:

$$n_j > n_i \tag{21}$$

Now the most probable and average value of t_0 is $32\,T$. The distribution has a very small tail corresponding to the solutions similar to this from Figure 1 when not the first host is empty at the end. The largest observed value of t_0 was $38\,T$.

Please note that Equation (18) can be easily generalized to the case of nonidentical containers and heterogeneous nodes. It is simply the amount of the least dominant resource from Equation (2) after all containers waiting in the outgoing and incoming queues have been migrated. Similarly, Equations (20) and (21) also hold in a general case (cf., Section 6.3).

In the next section we will study the performance and scaling of the distributed algorithm proposed in a simulated cloud. It was not possible to do this experimentally due to the limitations of our setup. The analysis of the scaling of the number of parallel migrations with cloud size will allow us to estimate the time complexity of the algorithm.

5. Computer Simulation of a Large Cloud

To investigate scaling of the number of parallel migrations t_0/T with the number of nodes N let us now build a cellular automaton-like simulator of the cloud. It is our own tool, written in Python. Its main design goals are to reproduce the experimental results from the previous section and allow the study of our algorithm in an arbitrarily large cloud.

We will assume that all nodes have the same capacity n_0 and all containers are identical. Thus, the occupations of the nodes will be represented by an array of natural numbers n_i. This is the number of currently running containers executing the cloud application. Some of them are waiting in the Q_i migration queues. The algorithm works in a time loop with step T until the final condition Equation (17) is met and all migration queues are empty.

At each time step a loop over nodes is executed. For each node the pheromone value is calculated according to Equations (18) and (19). Than in another loop each of $n_i - Q_i$ containers decides with probability p_i if it wants to migrate to another node. If so, then it starts a search for a suitable node that matches the conditions Equations (20) and (21). If the search fails nothing happens. If the search is successful, then the destination host j is added to the migration queue Q_i and the counter R_j is incremented.

To model the finite network throughput we will assume that at each time step of the simulation only one container can leave a given node and that migration of each container takes time T. Thus, after the addition of the containers to the migration queues another loop over nodes is executed. For each node i the first value j is taken from the queue Q_i and the value of n_j is increased while decreasing the counter R_j.

We prepared an initial state in which all the cloud nodes are equally occupied:

$$n_i = n(0), \quad i = 1, \ \ldots, \ N \tag{22}$$

Three choices for $n(0)$ were considered: $n_0 - 1$, $n_0/2$, and 1. To guarantee that the final state given by Equation (17) will be reached the number of nodes N was chosen as a multiplicity of n_0. Thus, for $n_0 = 50$ the numerical experiment was performed for $N = 50, 100, \ldots, 1000$. Depending of the choice of $n(0)$ at the end of the simulation 2%, 50%, and 98% nodes are empty and the remaining 98%, 50%, and 2%—full.

For each value of N the simulation was repeated $10,000$ times. The output from the program is a histogram of the number of steps t_0/T needed to reach the final state Equation (17). Example results for $N = 1000$ and $n(0) = 49$ (2% nodes empty and 98% full at the end) and $n(0) = 25$ (50% nodes empty and 50% full at the end) are plotted in Figure 2.

If we assume that the containers can leave a node only sequentially, but arrive to a node in parallel, then an ideal algorithm would need only $n(0)$ steps to reach the final state Equation (17) from the initial state Equation (22). In the most realistic case, when 2% nodes are empty and can be switched off at the end our algorithm works only about two times

worse than that (the average value $\langle t_0 \rangle \simeq 82\,T$ and $n(0) = 49$). The case of 50% empty nodes is about 4 times worse ($\langle t_0 \rangle \simeq 108\,T$ and $n(0) = 25$). Notice however the long tail of the distribution. The least realistic case with 98% empty servers (not plotted to keep the figure more clear) is more than 40 times worse than the optimal one ($\langle t_0 \rangle \simeq 42\,T$ and $n(0) = 1$).

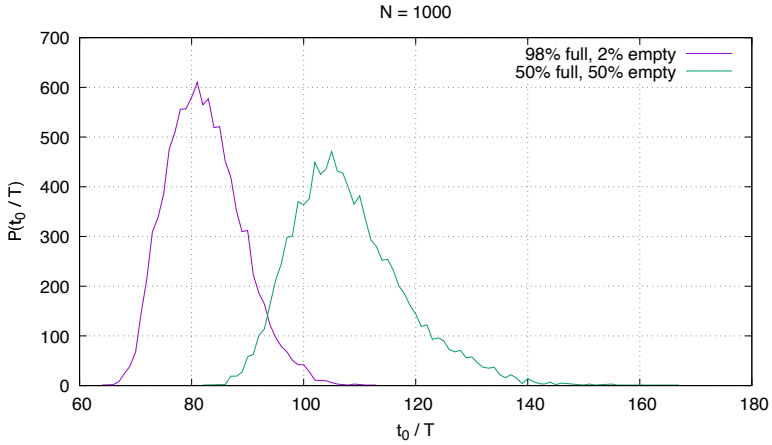

Figure 2. Distribution of times t_0 needed to reach equilibrium.

Analogous distributions were obtained for different values of N and the average values computed. The resulting average time needed to reach equilibrium t_0 versus cloud size N is plotted in Figure 3 using logarithmic scale on the horizontal axis. In both cases we obtain a straight line (this is also true in the case of $n(0) = 1$ not plotted here). Thus, $\langle t_0 \rangle \propto \log N$.

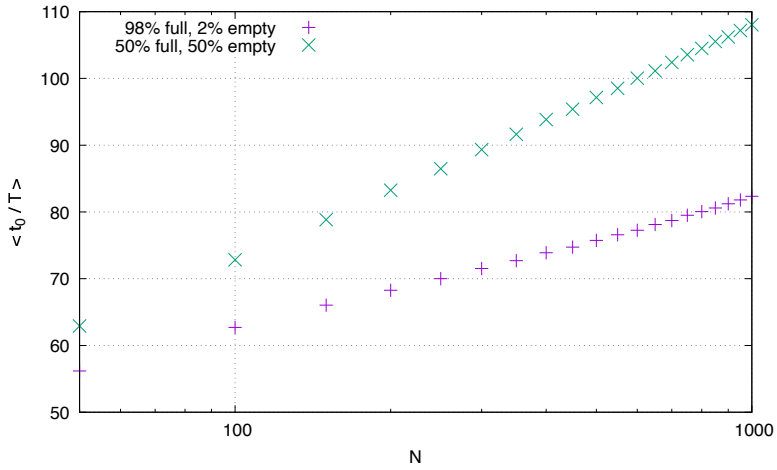

Figure 3. Average time needed to reach equilibrium $\langle t_0 \rangle$ versus cloud size N.

A very important component not accounted for in our numerical experiments is the time needed by a container to find a suitable node to migrate onto. In the simulation all the nodes were arranged in a ring and linear search was used starting at the node after the current node. Search is done in parallel by all nodes. Therefore the search time needed to be added to t_0 at each time step is proportional to N (the worst case is 50% empty, 50% full because the number of containers needed to be migrated is largest). Thus, the time-complexity of the distributed algorithm for scheduling containers to cloud nodes presented in this paper should be (at least) linear-logarithmic $O(N \log N)$.

Better search algorithms for Peer-to-Peer networks exist with $O(\log N)$ time complexity. One example is Chord [42]. If it would be possible to use such an Chord-like algorithm to find the node suitable for container migration, then the time complexity of our algorithm

would reduce to polylogarithmic one $O(\log^2 N)$. It would be better than known approximations for the classic bin-packing problem with linear or linear-logarithmic complexity (but our system is inherently parallel). This is a topic of ongoing research on our part.

6. Discussion

One of the major challenges that cloud providers face in the Infrastructure as a Service model is optimizing virtual machine placement on cloud nodes mostly within the context of minimizing power consumption of their data centers. The papers we cited in Section 1 dealt with this problem. However, containers are increasingly gaining popularity as a major deployment model in cloud environment specifically within the Containers as a Service cloud-computing model (that lies between the Infrastructure as a Service, and Platform as a Service models). In [43] the authors focused on improving the energy efficiency of servers for this new deployment model by proposing a framework being an extension of CloudSim [10,11] that consolidates containers on virtual machines. Later this framework has been generalized as ContainerCloudSim, which provides support for modeling and simulation of containerized cloud computing environments [44]. In this section we will compare our decentralized approach with results obtained by these authors.

The centralized framework proposed in [43,44] tackles the container consolidation problem in three stages. First, the situations in which container migration should be triggered are identified. Secondly, several containers to migrate in order to resolve the situation is selected. Finally, the migration destinations for the selected containers are found. Let us now discuss these stages in the following subsections.

6.1. Overloaded/Underloaded Node Identification

In the first stage of the framework described in [43], container migration is initiated if a host is identified as overloaded or underloaded considering a specific criteria. For this purpose, the host status module, which executes on each active host in the data center, checks the CPU use of the host every five minutes. Two static underload and overload thresholds are used for initiating the migrations.

Within our approach not only CPU, but all kinds of resources are taken into account. Each cloud node periodically calculates the dominant resource used $n_{\text{node}}^{(\text{dominant})}$ (cf., Equation (1)), and obtains the migration probability for the spread strategy $p_{\text{node}}^{(\text{spread})}$ (cf., Equation (3)). If it is positive, then the node is overloaded. Otherwise the node is underloaded. In this case the least dominant resource is calculated $n_{\text{node}}^{(\text{least})}$ (cf., Equation (2)), and the migration probability for the bin pack strategy is obtained $p_{\text{node}}^{(\text{spread})}$ (cf., Equation (4)). Thus, the final value of the container migration probability on a given node reads as:

$$p_{\text{node}} = \begin{cases} p_{\text{node}}^{(\text{spread})} & \text{if} \quad p_{\text{node}}^{(\text{spread})} > 0 \\ p_{\text{node}}^{(\text{binpack})} & \text{if} \quad p_{\text{node}}^{(\text{spread})} \leq 0 \end{cases} \tag{23}$$

Please note that in order to model an arbitrary overload and underload thresholds the node capacities in Equations (3) and (4) can be multiplied by respective factors <1. For example the authors [43] report that with 70% and 80% as underload and overload thresholds worked best for them.

6.2. Container Selection

Several container selection algorithms have been tested in [43]. The most promising turned out the one that selects for migration the containers with the most CPU use. Within our approach the migration decision is based on the value of the migration probability. It is given by Equation (23) and takes into account the usage of all resources on a node, and not only CPU.

It is known that non-deterministic, or randomized, algorithms for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers typically

improve upon the quality of their deterministic counterparts [9]. Within our approach each container periodically generates a random number $0 < r_{\text{container}} < 1$ and compares it to the migration probability:

$$r_{\text{container}} < p_{\text{node}} \tag{24}$$

It has been reported in [44] that adding the containers with the maximum CPU use to the migration list results in less container migrations. So far we considered the migration probability p_{node} to be identical for all containers on the node. However, it can be multiplied by a factor depending on a container resource usage. For example, to make the container with the most resource usage to migrate more likely the container migration probability $p_{\text{container}}$ could be defined as:

$$p_{\text{container}} = \frac{n_{\text{container}}}{n_{\text{max}}} p_{\text{node}} \tag{25}$$

where $n_{\text{container}}$ is the dominant/least dominant resource usage of a given container (depending on the spread or bin pack strategy chosen), and n_{max} is the maximum of this value for all containers. The validity of Equation (25) can be checked by more realistic experiments similar to these from Section 4, but involving a larger cloud and heterogeneous containers corresponding to a real application. We plan to do this in future work.

6.3. Destination Node Selection

In the system proposed by [43] the list of containers that are required to be migrated is sent to the consolidation manager module. The consolidation manager is installed on a separate node of the cloud and can be replicated to avoid a single point of failure. It decides about the new placement of containers and sends requests to provision resources to the destination host [44]. Initially, the First-Fit algorithm is applied to select the destination node whose capacity matches a predefined percentile of CPU workload of the container. Additionally, a correlation aware host selection algorithm is also used. It matches the destination host workload history with the container workload history. If the container workload is not correlated with the host load, there is less probability of that container causing the host to become overloaded. It was shown that this destination host selection policy reduces the energy consumption in most of the scenarios [43,44].

In our system proposed in this paper also two conditions for the destination node selection are used. However, they take into account all kinds of resources, not only CPU workload. First we require that *after* the container migration the value of the repulsive pheromone from Equation (3) remains non-positive (cf., Equation (20)):

$$p^{(\text{spread})}_{\text{destination}} \leq 0 \tag{26}$$

Please note that after the migration the dominant resource kind on the destination node may change. The condition above is used for both spread and bin pack strategies. (cf., Equation (21)). In addition, to ensure the fast convergence of the bin pack strategy, we require, that destination node is chosen in such way, that the value of the repulsive pheromone from Equation (4) *after* the migration is smaller than the value of the repulsive pheromone on the initial host *i before* the migration (cf., Equation (21)):

$$p^{(\text{binpack})}_{\text{destination}} < p^{(\text{binpack})}_{\text{initial}} \tag{27}$$

Please note that the condition above can be fulfilled even if the system starts with equally prepared nodes (cf., Equation (22)).

6.4. Container Migration

The main advantage of our decentralized system comes from its inherently parallel nature. In a centralized system [43,44] container migrations are done sequentially by the

consolidation manager. Thus, if the number of migrations needed to reach the desired state of the cloud is M, then the time needed to migrate all the containers is $t_0 = O(M)$.

The minimal number of container migrations M needed to reach the final state given by Equation (17) from the initial state given by Equation (22) is proportional to the cloud size N (which was chosen to be the multiplicity of n_0):

$$M = \left(N - \frac{N}{n_0} n(0) \right) n(0) \tag{28}$$

In our simulations described in Section 5 and the total number of migrations was at average 80% larger than the value given by Equation (28). However, the migrations were done in parallel by different nodes of the cloud and the time needed to migrate all the containers was only $t_0 = O(\log M)$.

Second advantage is that in the system proposed in this paper there is no limit on the container lifetime. The average lifespan of a container is short and becoming shorter. In a recent report a majority of container instances was shown to life only around for five minutes or less (https://sysdig.com/blog/sysdig-2019-container-usage-report/, accessed on 26 June 2021). In the case of the centralized system described in [43,44], if a container dies after its data are send to the consolidation manager, its copy will be still started on the destination node. This can lead to the waste of cloud resources.

And the third advantage is that there is no shutdown or startup requirement on the application running inside virtualization containers. The simulations presented in [43,44] assumed that containers are shutdown and started on the nodes when they are need to be moved. This introduces two time delays. One is the time needed to shutdown the container as the application inside could use some shared storage and it could be in process of writing some data into it. Then it has to be launched on the destination node. However, some applications can take a few seconds to be ready to work after a container starts. Within our approach neither shutdown nor startup are necessary because the container is being live-migrated to the destination host, and thus the state of the application running inside it is not lost.

6.5. Algorithm Convergence and Correctness

Using the mathematical model presented in Section 3 the convergence and asymptotic correctness of the proposed algorithm with random jumps has been proven rigorously in the case of spread strategy (cf., Equation (3)) and arbitrarily large number of nodes [38]. In the case of bin pack strategy (cf., Equation (9)) the rigorous proof was given in Section 3 for two nodes only. For three (and larger number of nodes) we studied only the stability of the stationary solutions. We do not know know an analytic solution of the system of coupled nonlinear differential Equations (5) in this case. However, we solved them numerically using Mathematica for three nodes and randomly chosen initial conditions. Different stationary solutions were always asymptotically reached depending on the initial state. However, only some of them represent the correct final configurations of the cloud (cf., Equation (17)).

Of course the mathematical proof does not guarantee convergence in a real system. For example, in [41] we studied experimentally a system of two nodes only implementing the spread strategy. It was shown that some container number oscillations not predicted by a mathematical model can occur. However, they can be suppressed by creating migration queues and modifying the resource usage on the cloud nodes seen by the algorithm by the resource usage of the containers waiting to be migrated (cf., Equation (18)). Our experiments indicate that this is also the case for the bin pack strategy studied in this paper.

Next, in Section 4 the bin pack strategy was studied experimentally in a three-node cluster. For an initial state given by Equation (16) 10,000 runs of the experiment were performed. The cloud always converged to the desired final state with one empty node (cf., Equation (8)). For a larger cloud and the initial state of the given by Equation (22) simulations similar to these from Section 5 were conducted. They also indicate that for each

of 10,000 runs, one node can be emptied in a linear time. This node can than be switched off, and the procedure repeated until all nodes that can be empty are empty. Thus, such a system has a quadratic time complexity.

Please note that the experimental initial state Equation (16) of the three nodes was chosen in such a way that the unwanted final state Equation (15) is not reached. To prevent this for an arbitrary initial state a condition Equation (26) for the destination host selection needs to be introduced. It ensures the correctness of the algorithm but its time complexity is still quadratic.

For random container jumps it is not possible to achieve a situation in which more then one node is empty. The containers keep coming back to the empty nodes and the algorithm never finishes. This situation can be resolved by adding the second condition on the destination node selection Equation (27). The experiments presented in Section 4 indicate a 5 times speedup of the algorithm with respect to random choice strategy. This improved algorithm has also been studied extensively using the cloud simulator from Section 5. The time complexity in this case is found to be logarithmic.

Each numerical experiment was repeated 10,000 times, 20 different cloud sizes, and 3 values for the empty/full node ratio were considered. In total 600,000 runs of the algorithm were performed—it always reached the desired final state Equation (17). This is of course not a rigorous proof, but a fairly good indication that the bin pack algorithm converges and gives the correct result. Please note that the simulator from Section 5 was designed to reproduce the experimental results from Section 4. Therefore, we believe that also the experiments conducted in a large cloud will converge to the desired final configuration.

Let us now go back to the initial form of the repulsive "pheromone" for the bin pack strategy given by Equation (4). In this case the numerical solutions of the system of Equation (5) randomly switch between the unstable stationary solutions Equation (8) due to numerical noise. Similarly, neither in the experiment nor in the simulation convergence to one empty node is observed. However, the addition of conditions Equation (27) and Equation (26) changes the situation drastically. Now, the simulator converges to the desired final state Equation (17) in logarithmic time. It is interesting to observe that such a system converges faster than Equation (9) for a small cloud. For a large 1000 node cloud it is however about 5% slower.

6.6. Threats to Validity

The experiments described in Section 4 were conducted using OpenVZ 7 container-based virtualization for Linux. In the previous versions of OpenVZ, most operations performed during migration were done in the kernel space. However, the release used by us is based on unmodified Red Hat Enterprise Linux 7 kernel, and container live migration is done via CRIU and P.Haul. These are standard userspace packages available on any modern Linux system. The main difference between using OpenVZ and Docker is a different container runtime. However, we believe that this should have no impact on the experimental results presented in this paper and conclusions drawn from them. Container technology is rapidly advancing. In our future experiments we plan to use runC (https://www.redhat.com/en/blog/container-live-migration-using-runc-and-criu, accessed on 26 June 2021) or podman (https://www.redhat.com/en/blog/container-migration-podman-rhel, accessed on 26 June 2021) live migration. This was not possible for the current experiment, because we wanted the containers to save logs for debug purposes. At the time this paper is written, stateful container migration only works with containers, which do not change their file system.

6.7. Possible Network Impact

The system proposed in this paper has two network impacts. The first is the cost of container migrations. In a recent paper [22] it has been proposed to migrate only the runtime state of the container, and not its base image. This reduces the migration cost substantially. Another impact is the cost of finding the suitable host to migrate to. This

search can be done by a P2P network of REST services started on cloud nodes (single service per node). However, in our opinion, this cost should be negligible with respect to the cost of overlay networks used by application containers. It has been reported that containers itself show a performance impact of about 10% to 20%. The impact of overlay networks is even worse, and reaches about 30% to 70% [45].

7. Conclusions

In summary, a swarm-like algorithm for virtualization container scheduling in the cloud using live migration was proposed. Each cloud node is assigned a "pheromone" value, which is calculated taking into account the resource usage of containers that are scheduled for execution on this node, as well as the node capacity. This value is evaluated on set time intervals by each container, which—when necessary—asks the host to migrate it to another execution node. Migration is possible only to the nodes with an attractive "pheromone". Finally, after several periods, the system converges to optimal load, providing a decentralized method to establish a desired configuration of the cloud.

Two main scheduling strategies were considered. The spread-like strategy applied to the microservice-based application running in the cloud causes virtualization containers to disperse evenly on the cloud nodes. This approach minimizes the risk of application failure when a node fails (because of the small number of containers on it). The bin pack-like strategy applied to the virtualization containers running in the cloud causes the application to run on as few servers as possible allowing savings of electrical energy powering the cloud cluster (because unused servers may be turned off). The cloud administrator using the system proposed can easily switch between these strategies by simply sending a single threshold parameter to cloud nodes using, e.g., the gossip protocol [46].

Usually, swarm models use a large number of unintelligent actors in order to produce complex global behaviors. It was shown that in the case that the bin pack strategy local (i.e., limited to the cloud node on which they are running) interactions between the containers are not sufficient. Some intelligence needs to be added to the swarm. It turns out that in order to yield a satisfactory performance of the algorithm a search for a suitable destination node is necessary.

The performance of the algorithm proposed has been studied using a cellular automaton-like cloud simulator. It was shown that depending if the search time is included or not the system can reach an optimal configuration in a polylogarithmic or logarithmic time. This result holds also in the case of the spread strategy (where local interactions are sufficient) and is better than existing container orchestrators that require at least linear time.

Our system uses live migration of containers (which is difficult to implement in centralized systems because all changes to the cloud configuration must be stored in a central database) and allows for the redistribution of already running containers. There are no master nodes, which need to be replicated. Each container schedules itself. This yields in a better failure tolerance. There is no lower limit on the container lifetime and no need to shutdown and restart containers in order to change their location in the cloud.

Please note that the algorithm proposed in this paper is general, and in principle could be used for virtual machine management. However, adding a mobile agent process requires changing a virtual machine setting. This is not always possible. In the case of containers both the application container and the mobile agent container can be combined together in a pod without the need for modifying them.

In the long-run, we believe we will see a move towards bin pack style scheduling for containers in the cloud in order to reduce hosting costs. Using stateless microservices where possible can mitigate the risk of service downtime—such services can be automatically and quickly migrated to new hosts with minimal disruption to availability. However, if one uses bin pack today, he needs to be aware of how to deal with co-scheduling of containers, i.e., how to ensure two or more containers are always run on the same host. In our approach this could be done by appropriately introducing attractive "pheromone" values. Such an analysis cannot be made using a cellular automaton model studied in this paper. It requires

a real cloud application and a real cloud to analyze its performance experimentally. This will be the topic of our future work.

In this paper, only swarms formed by containers were studied. An important point left out for a future paper was building a P2P network from the supplementary REST services launched on each node. As described at the end of Section 5 they will implement a Chord-like search algorithm, aiding the containers in finding a suitable migration destination. This requires a fairly large experimental setup using modern container technology. Last but not least, also other research directions mentioned in Section 6 look interesting: the experimental study of different migration priorities for containers of different sizes and resource requirements, and the proper analysis of the network impact of the proposed scheme.

Author Contributions: Algorithm, M.R. and G.D.; model, M.R.; experiment, G.D.; simulation, M.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
API	Application Programming Interface
CPU	Central Processing Unit
CRIU	Checkpoint/Restore In Userspace
CaaS	Containers as a Service
DRF	Dominant Resource Fairness
DVMS	Distributed Virtual Machine Scheduler
HDD	Hard Disk Drive
HaaS	Hardware as a Service
IBM	International Business Machines Corporation
IaaS	Infrastructure as a Service
IoT	Internet-of-Things
KVM	Kernel-based Virtual Machine
NP	Nondeterministic Polynomial
P2P	Peer-to-Peer
PaaS	Platform as a Service
RAM	Random Access Memory
REST	Representational State Transfer
SOMA	Self-Organizing Migrating Algorithm
SaaS	Software as a Service

Appendix A

Table A1. Notations used in this paper.

M	number of container migrations needed to reach the desired state of the cloud
N	number of cloud nodes
Q	length of the emigration queue on a node
R	length of the immigration queue on a node
T	container migration time
$\langle\rangle$	average value
i	cloud node number ($1 \leq i \leq N$)
j	cloud node number ($1 \leq j \leq N$)
k	resource kind (CPU, RAM, HDD, etc.)
n	dominant/least dominant resource usage on a node
$n^{(k)}$	resource usage of a given kind on a node

Table A1. *Cont.*

n_0	dominant/least dominant resource capacity of a node
$n_0^{(k)}$	capacity of a node corresponding to resources of a given kind
$n_{\max}$	maximal dominant/least dominant resource usage from all containers on a node
$n_{\min}$	minimal least dominant resource usage from all containers on a node
p	container migration probability/"pheromone" value
r	random number ($0 < r < 1$)
t	time
t_0	time needed to reach the desired state of the cloud

Please note that indices i and j are omitted whenever possible to make the equations more readable. In the case of identical containers n and n_0 are simply their numbers, and $n_{\min} = n_{\max} = 1$.

Appendix B

Algorithm A1: Mobile Agent Code Run by Each Container

```
1:  loop
2:    p ← GetRepulsivePheromoneFromTheHost()
3:    r ← GenerateRandomNumber()
4:    if r < p then
5:      d ← AskTheHostToFindASuitableDestinationNode(requirements)
6:      if d then
7:        AskTheHostToMigrateMeToNode(d)
8:        WaitUntilMigrationIsComplete()
9:      end if
10:   end if
11:   SleepForSomeTime()
12: end loop
```

Each cloud node periodically checks the resource usage of all containers running on it and computes the repulsive "pheromone" value. The cloud nodes form a Peer-to-Peer network, which allows them to aid the containers in choosing a suitable destination node with an attractive "pheromone". Container requirements may define additional affinity constraints specifying which containers need to be on the same node.

References

1. Zhang, Q.; Cheng, L.; Boutaba, R. Cloud computing: State-of-the-art and research challenges. *J. Internet Serv. Appl.* **2010**, *1*, 7–18. [CrossRef]
2. Scheepers, M.J. Virtualization and containerization of application infrastructure: A comparison. In Proceedings of the 21st Twente Student Conference on IT, Enschede, The Netherlands, 23 June 2014; pp. 1–7.
3. Brewer, E.A. Kubernetes and the path to cloud native. In Proceedings of the Sixth ACM Symposium on Cloud Computing, Kohala Coast, HI, USA, 27–29 August 2015; p. 167.
4. Kratzke, N. About the complexity to transfer cloud applications at runtime and how container platforms can contribute? In Proceedings of the International Conference on Cloud Computing and Services Science, Porto, Portugal, 24–26 April 2017; pp. 19–45.
5. Kratzke, N. A brief history of cloud application architectures. *Appl. Sci.* **2018**, *8*, 1368. [CrossRef]
6. Naik, N. Building a virtual system of systems using Docker Swarm in multiple clouds. In Proceedings of the 2016 IEEE International Symposium on Systems Engineering (ISSE), Edinburgh, UK, 3–5 October 2016; pp. 1–3.
7. Verma, A.; Ahuja, P.; Neogi, A. pMapper: Power and migration cost aware application placement in virtualized systems. In Proceedings of the ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, Leuven, Belgium, 1–5 December 2008; pp. 243–264.
8. Hermenier, F.; Lorca, X.; Menaud, J.M.; Muller, G.; Lawall, J. Entropy: A consolidation manager for clusters. In Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, Washington, DC, USA, 11–13 March 2009; pp. 41–50.

9. Beloglazov, A.; Buyya, R. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurr. Comput. Pract. Exp.* **2012**, *24*, 1397–1420. [CrossRef]
10. Calheiros, R.N.; Ranjan, R.; De Rose, C.A.; Buyya, R. Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services. *arXiv* **2009**, arXiv:0903.2525.
11. Calheiros, R.N.; Ranjan, R.; Beloglazov, A.; De Rose, C.A.; Buyya, R. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exp.* **2011**, *41*, 23–50. [CrossRef]
12. Hermenier, F.; Demassey, S.; Lorca, X. Bin repacking scheduling in virtualized datacenters. In Proceedings of the International Conference on Principles and Practice of Constraint Programming, Perugia, Italy, 12–16 September 2011; pp. 27–41.
13. Verma, A.; Pedrosa, L.; Korupolu, M.; Oppenheimer, D.; Tune, E.; Wilkes, J. Large-scale cluster management at Google with Borg. In Proceedings of the Tenth European Conference on Computer Systems, Bordeaux, France, 21–24 April 2015; pp. 1–17.
14. Ousterhout, K.; Wendell, P.; Zaharia, M.; Stoica, I. Sparrow: Distributed, low latency scheduling. In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, Farmington, PA, USA, 3–6 November 2013; pp. 69–84.
15. Quesnel, F.; Lèbre, A.; Südholt, M. Cooperative and reactive scheduling in large-scale virtualized platforms with DVMS. *Concurr. Comput. Pract. Exp.* **2013**, *25*, 1643–1655. [CrossRef]
16. Gog, I.; Schwarzkopf, M.; Gleave, A.; Watson, R.N.; Hand, S. Firmament: Fast, centralized cluster scheduling at scale. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), Savannah, GA, USA, 2–4 November 2016; pp. 99–115.
17. Jamshidi, P.; Pahl, C.; Mendonça, N.C.; Lewis, J.; Tilkov, S. Microservices: The journey so far and challenges ahead. *IEEE Softw.* **2018**, *35*, 24–35. [CrossRef]
18. De Donno, M.; Tange, K.; Dragoni, N. Foundations and evolution of modern computing paradigms: Cloud, IoT, edge, and fog. *IEEE Access* **2019**, *7*, 150936–150948. [CrossRef]
19. Taherizadeh, S.; Stankovski, V.; Grobelnik, M. A capillary computing architecture for dynamic internet of things: Orchestration of microservices from edge devices to fog and cloud providers. *Sensors* **2018**, *18*, 2938. [CrossRef]
20. Kumar, V.; Laghari, A.A.; Karim, S.; Shakir, M.; Brohi, A.A. Comparison of fog computing & cloud computing. *Int. J. Math. Sci. Comput.* **2019**, *1*, 31–41.
21. Qu, Q.; Xu, R.; Nikouei, S.Y.; Chen, Y. An Experimental Study on Microservices based Edge Computing Platforms. In Proceedings of the IEEE INFOCOM 2020—IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Toronto, ON, Canada, 6–9 July 2020; pp. 836–841.
22. Puliafito, C.; Vallati, C.; Mingozzi, E.; Merlino, G.; Longo, F.; Puliafito, A. Container migration in the fog: A performance evaluation. *Sensors* **2019**, *19*, 1488. [CrossRef] [PubMed]
23. Baburao, D.; Pavankumar, T.; Prabhu, C. Migration of Containerized Microservices in the Fog Computing Environment. *Turk. J. Physiother. Rehabil.* **2021**, *32*, 3.
24. Fard, H.M.; Prodan, R.; Wolf, F. A container-driven approach for resource provisioning in edge-fog cloud. In Proceedings of the International Symposium on Algorithmic Aspects of Cloud Computing, Munich, Germany, 10 September 2019; pp. 59–76.
25. Rusek, M.; Dwornicki, G.; Orłowski, A. A decentralized system for load balancing of containerized microservices in the cloud. In Proceedings of the International Conference on Systems Science, Wroclaw, Poland, 7–9 September 2016; pp. 142–152.
26. Hoenisch, P.; Weber, I.; Schulte, S.; Zhu, L.; Fekete, A. Four-fold auto-scaling on a contemporary deployment platform using docker containers. In Proceedings of the International Conference on Service-Oriented Computing, Goa, India, 16–19 November 2015; pp. 316–323.
27. Zelinka, I. SOMA—Self-organizing migrating algorithm. In *New Optimization Techniques in Engineering*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 167–217.
28. Payton, D.; Estkowski, R.; Howard, M. Progress in pheromone robotics. *Intell. Auton. Syst.* **2002**, *7*, 256–264.
29. Payton, D.; Estkowski, R.; Howard, M. Compound behaviors in pheromone robotics. *Robot. Auton. Syst.* **2003**, *44*, 229–240. [CrossRef]
30. Cheng, J.; Cheng, W.; Nagpal, R. Robust and self-repairing formation control for swarms of mobile agents. *AAAI* **2005**, *5*, 59–64.
31. Berman, S.; Halász, A.; Kumar, V.; Pratt, S. Bio-inspired group behaviors for the deployment of a swarm of robots to multiple destinations. In Proceedings of the 2007 IEEE International Conference on Robotics and Automation, Rome, Italy, 10–14 April 2007; pp. 2318–2323.
32. Cheah, C.C.; Hou, S.P.; Slotine, J.J.E. Region-based shape control for a swarm of robots. *Automatica* **2009**, *45*, 2406–2411. [CrossRef]
33. Rubenstein, M. *Self-Assembly and Self-Healing for Robotic Collectives*; University of Southern California: Los Angeles, CA, USA, 2010.
34. Rubenstein, M.; Cornejo, A.; Nagpal, R. Programmable self-assembly in a thousand-robot swarm. *Science* **2014**, *345*, 795–799. [CrossRef]
35. Oh, H.; Shiraz, A.R.; Jin, Y. Morphogen diffusion algorithms for tracking and herding using a swarm of kilobots. *Soft Comput.* **2018**, *22*, 1833–1844. [CrossRef]
36. Karwowski, W.; Rusek, M.; Dwornicki, G.; Orłowski, A. Swarm based system for management of containerized microservices in a cloud consisting of heterogeneous servers. In Proceedings of the International Conference on Information Systems Architecture and Technology, Szklarska Poreba, Poland, 17–19 September 2017; pp. 262–271.
37. Ghodsi, A.; Zaharia, M.; Hindman, B.; Konwinski, A.; Shenker, S.; Stoica, I. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. *NSDI* **2011**, *11*, 24.

38. Rusek, M.; Landmesser, J. Time Complexity of an Distributed Algorithm for Load Balancing of Microservice-oriented Applications in the Cloud. *ITM Web Conf.* **2018**, *21*, 18. [CrossRef]
39. Smith, J.M. A survey of process migration mechanisms. *ACM SIGOPS Oper. Syst. Rev.* **1988**, *22*, 28–40. [CrossRef]
40. Mirkin, A.; Kuznetsov, A.; Kolyshkin, K. Containers checkpointing and live migration. In Proceedings of the Linux Symposium, Ottawa, ON, Canada, 23–26 July 2008; pp. 85–90.
41. Rusek, M.; Dwornicki, G.; Orłowski, A. Swarm of Mobile Virtualization Containers. In Proceedings of the 36th International Conference on Information Systems Architecture and Technology–ISAT 2015–Part III, Karpacz, Poland, 20–22 September 2015; pp. 75–85.
42. Stoica, I.; Morris, R.; Karger, D.; Kaashoek, M.F.; Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Comput. Commun. Rev.* **2001**, *31*, 149–160. [CrossRef]
43. Piraghaj, S.F.; Dastjerdi, A.V.; Calheiros, R.N.; Buyya, R. A framework and algorithm for energy efficient container consolidation in cloud data centers. In Proceedings of the 2015 IEEE International Conference on Data Science and Data Intensive Systems, Sydney, Australia, 11–13 December 2015; pp. 368–375.
44. Piraghaj, S.F.; Dastjerdi, A.V.; Calheiros, R.N.; Buyya, R. ContainerCloudSim: An environment for modeling and simulation of containers in cloud data centers. *Softw. Pract. Exp.* **2017**, *47*, 505–521. [CrossRef]
45. Kratzke, N. About microservices, containers and their underestimated impact on network performance. *arXiv* **2017**, arXiv:1710.04049.
46. Eugster, P.T.; Guerraoui, R.; Kermarrec, A.M.; Massoulié, L. Epidemic information dissemination in distributed systems. *Computer* **2004**, *37*, 60–67. [CrossRef]