Time Complexity of an Distributed Algorithm for Load Balancing of Microservice-oriented Applications in the Cloud

Marian Rusek[1,*] and *Joanna* Landmesser[2]

[1,2]Warsaw University of Life Sciences – SGGW, Nowoursynowska 159, 02-776 Warsaw, Poland

Abstract. Microservice architecture is a relatively new cloud application design pattern. Each microservice has a single responsibility in terms of functional requirement, and that can be managed independently from other microservices. This is done using automated cloud orchestration systems. In this paper we analyze the time complexity of an simple swarm-like decentralized load balancing algorithm for microservices running inside OpenVZ virtualization containers. We show that it can offer performance improvements with respect to the existing centralized container orchestration systems.

1 Introduction

Monolithic enterprise systems are difficult to scale, difficult to understand and difficult to maintain. Microservices-based architecture is free of these problems [1]. Designed with cloud-based applications in mind it advocates creating a system from a collection of small, isolated microservices, each of which owns their data and uses lightweight HTTP mechanisms for communication with other microservices. These microservices integrate with each other in order to form a cohesive system that is more flexible, scalable and resilient to failure than a corresponding monolithic system. In this way the accidental complexity is shifted from inside of an monolithic application into the cloud infrastructure. Microservices are deployable by fully automated cloud machinery, but there is still a necessity for centralized monitoring and management of an microservces based application. Examples of such microservices orchestration tools are Google Kubernetes [2], Apache Mesos [3], and Docker Swarm [4].

Virtualization containers are ideal for practical realization of the concept of microservices [5]. Virtualization container is a group of processes isolated from the rest of the system. Different containers running on the same machine share the host's operating system kernel. Recent interest in virtual containers is associated with their performance. Public cloud virtual machines have to boot a full operating system and for every time it takes up to several minutes. In contrast to this the startup time for a container is around a second. This technology is not new but before Docker there was no common interface for container management and there was no standard format of how container is described in

* Corresponding author: marian_rusek@sggw.pl

the system. All three orchestration systems mentioned above work with Docker containers. The role of orchestration and scheduling within these container platforms is to match microservices-based application containers to cloud servers resources.

The actual realization of a microservices-based application in the cloud requires that we instantiate and place virtualization containers responsible for individual microservices on real machines forming the cloud. Container live migration allows a user to start a microservice on any node of the cloud after which it can transparently move to other nodes to make efficient use of resources. The basic idea is that overall system performance can be improved if microservices are moved from heavily loaded to lightly loaded machines. However instead of offloading machines we can imagine that code is moved to make sure that a machine is sufficiently loaded. For example, migrating complete virtual machines to lightly loaded machines in order to minimize the total number of nodes being used is a common practice in optimizing energy usage in data centers [6]. Another advantage of live migration is the ability to put the code processing the data close to where the data reside. This allows minimizing communication what is an important issue in today's distributed cloud environments [7]. In this paper we focus on the first from these scenarios (load balancing).

Note, that Docker compatible container systems can not provide live migration. Docker was created for different purpose, an automated application deployment, but there is commonly used, container-based virtualization software, Parallels Containers, which supports live migration [8]. Thus the experimental results discussed in our previous papers were obtained using OpenVZ, an open source version of Parallels Containers. Moreover all the existing container orchestration systems (Google Kubernetes, Apache Mesos, and Docker Swarm) are monolithic applications running as daemons on dedicated nodes of the cloud. They orchestrate containers in a centralized fashion. Usually decentralized orchestration systems offer performance improvements [9]. For example, decentralized orchestration of composite web services yields increased throughput, better scalability, and lower response time. In connection to this, in this paper we analyze the performance of a completely decentralized microservice management system based on swarm algorithm.

This paper is organized as follows: In Sec. 2 the distributed algorithm for load balancing of containerized microservices in the cloud proposed in our previous paper [10] is recalled. In that previous paper a "cloud" consisting of 18 servers only was studied experimentally. In Sec. 3 a mathematical model is introduced which allows us to investigate arbitrarily large clouds and analyze the time complexity of our algorithm. In Sec. 4 numerical simulations for 1000 servers and 50000 containers are also presented. We finish with a summary and some conclusions in Sec. 5.

2 Distributed Algorithm

The question of the optimal distribution of the containers on the hosts can have multiple variants. One of them is the uniform distribution of the containers on the hosts. This scenario is the following: we have N hosts (cloud servers) loaded with virtualization containers (containing microservices). At the start containers are arranged in a way that there is not the same number of them n_i on i-th host. Uniform distribution means that the targeted number of containers on every host is equal to n_0, and the goal of the algorithm is to achieve this state of balance. A decentralized algorithm for controlling the uniform distribution of tasks between nodes of a simple computational cloud was presented in our previous paper. In that approach, each host is described by a pheromone p_i which can be either repulsive $0 < p_i < 1$ or attractive $p_i < 0$. Attractiveness or repulsiveness is associated with the number of containers already n_i on the host and the number of containers queued for migration in the kernel queue Q_i:

$$p_i = \frac{n_i - Q_i - n_0}{n_i - Q_i}, \quad i = 1,\ldots,N \tag{1}$$

Note, that during migration the container's files are present on both source and destination hosts. Therefore the total number of containers calculated as sum of n_i over all hosts is not conserved. To prevent the containers from being counted twice, what in Eq. (1), we subtracted from the number of containers n_i present in the file system of the i-th hosts the number of containers Q_i queued for migration from this host. In this way the migration process of a container appears to be instantaneous to its neighboring containers on the same host. As shown in our previous paper [11] this prevents the unstable behavior of the algorithm without this subtraction.

In addition to cloud application microservice each container runs an additional process implementing it's swarm-like mobile agent intelligence. It executes the following pseudocode [11]:

```
1: loop
2:   Compute the pheromone value p of the current host.
3:   Generate a random number 0 < r < 1.
4:   if r < p
5:     repeat
6:       Randomly choose a new host.
7:       Get the pheromone value of the chosen host.
8:     until chosen host has an attractive pheromone
9:     Ask the current host to migrate to chosen host.
10:    Wait until migration to another host is complete.
11:  end if
12: end loop
```

Thus the pheromone value p_{ij} of the host is equal to migration probability of a container.

In the following it will be assumed for simplicity that the containers are identical and have almost the same memory, disk, and processor requirements. An analogous assumption will be made for the hosts: the memory, disk, and processing performance of each host is the same.

3 Mathematical Model

The distributing of a robot swarm among multiple tasks can be described using delay differential equations [12]. Let us write a similar system of equations for the swarm of containers from Sec. 2. The main difference is that in [12] the migration probabilities between hosts are constant, and it can be shown that the system has a unique, stable equilibrium point. In contrast we dynamically adjust the migration probabilities (see Eq. (1)) so the system reaches not some equilibrium point, but an equilibrium point that is interesting to us (i.e., $n_i = n_0$).

Let us assume that it takes a time T to migrate a container between any pair of hosts. The change in time of the number of containers Q_i queued for migration on the i-th host is due to new containers entering it with probability p_i and containers leaving it due to migration to other hosts. Thus

$$\frac{dQ_i(t)}{dt} = p_i(t)n_i'(t) - Q_i(t-T), \quad i = 1,\ldots,N \tag{2}$$

where $n_i^{'}(t) = n_i(t) - Q_i(t),\ i = 1,\ldots,N$ is the number of containers still undecided to migrate, i.e., generating random numbers *r* and comparing them to p_i in the loop from Sec. 2. The change in this number is due to containers entering the migration queue on the same host and migrating randomly from other hosts with probability 1/(*N* - 1):

$$\frac{dn_i^{'}(t)}{dt} = -p_i(t)n_i^{'}(t) + \frac{1}{N-1}\sum_{j\neq i} Q_j(t-T), \quad i = 1,\ldots,N \tag{3}$$

A finite number of containers will usually be en route between hosts, but it follows from Eqs. (2) and (3) that the total number of containers is conserved: $\frac{d}{dt}\sum_i n_j(t) = 0$.

Note, that the delayed differential Eqs. (2) and (3) model can not take into account the finite network bandwidth. The containers migrate during time T but infinitely many of them can migrate in parallel. This will be corrected in the numerical simulation presented in the next section.

Usually a solution of delayed differential equations is not possible using elementary functions. Therefore in this section we will limit ourselves to the solution of a simpler linear model. It assumes that containers instantaneously migrate from one host to another. The system of equations governing the dynamics of the system reads as:

$$\frac{dn_i^{'}(t)}{dt} = -p_i(t)n_i^{'}(t) + \frac{1}{N-1}\sum_{j\neq i} p_j(t)n_j^{'}(t), \quad i = 1,\ldots,N \tag{4}$$

After substituting Eq. (1) and introducing a new notation $n_i^{''}(t) = n_i^{'}(t) - n_0,\ i = 1,\ldots,N$, Eqs. (4) reduce to the system of linear equations:

$$\frac{dn_i^{''}(t)}{dt} = -n_i^{''}(t) + \frac{1}{N-1}\sum_{j\neq i} n_j^{''}(t), \quad i = 1,\ldots,N \tag{5}$$

which can be easily solved using diagonalization. There are N - 1 eigenvectors

$$[-1,1,0,\ldots,0], \quad [-1,0,1,\ldots,0], \quad \ldots \quad ,[-1,0,0,\ldots,0,1] \tag{6}$$

corresponding to eigenvalues - *N*/(*N* - 1), and one eigenvector [1, ..., 1] corresponding to eigenvalue 0. Thus all solutions decay exponentially in time. Interestingly, a system of two hosts decays twice as fast as a system of a large number of hosts.

Also as opposed to the algorithm from Sec. 2 the choice of the destination host in Eqs. (5) (and (3)) is random. However this choice is less effective than the choice of the host with an attractive pheromone. Therefore the solution of Eq. (6) should predict a correct upper bound on the pessimistic time complexity of the swarm algorithm from Sec. 2. The initial state of the system $n_i^{''}(0),\ i = 1,\ldots,N$, can be expanded into eigenvectors from Eq. (6). They then decay exponentially.

First we consider a limiting case of one empty host at the initial time:

$$\begin{aligned} &n_1(0) = 0 \\ &n_i(0) = n_0 + 1, \quad i = 2,\ldots,N, \\ &n_0 = N - 1 \end{aligned} \tag{7}$$

In order to reach the final state $n_i = n_0$ the number of N - 1 containers need to be migrated. It is readily seen that $n''(0) = [-(N-1), 1, \ldots, 1]$. This vector is equal to the sum of all eigenvectors from Eq. (6) corresponding to the same eigenvalue. Let us denote by τ the time for which $n_i''(\tau) < 1$. For $t > \tau$ it is already decided which containers should move where in order to bring the system into the desired equilibrium state $n_i(t_0) = n_0$. The time τ depends on the size of the cloud as log(N). Thus the time t_0 at which equilibrium is reached grows in the worst case linearly with N. Thus the time complexity of our distributed algorithm should be O(N) or better. We will check this prediction in the next section.

Interestingly the second limiting case of one non empty host at the initial time:

$$\begin{aligned} & n_1(0) = N \\ & n_i(0) = 0, \quad i = 2, \ldots, N, \\ & n_0 = 1 \end{aligned} \tag{8}$$

correspond to the same initial vector $n''(0) = [N-1, -1, \ldots, -1]$. And also N - 1 containers need to be migrated. In the next section using a more realistic numerical model, we will show that surprisingly this case behaves different from the previous one.

4 Numerical Simulation

In this section results given by a simple cellular automaton-like simulator are presented. The simulation operates on two arrays: n_i' and Q_i — thus the containers are identical and indistinguishable from each other. The simulator is single-threaded but nevertheless is able to reproduce the experimental distribution of stabilization times t0 for a system of 18 hosts presented in our previous paper [11]. At each time step all the active (i.e., not waiting in the migration queue) containers calculate their migration probability according to Eq. (1), decide to migrate or not to migrate, and chose their destination host with an attractive pheromone like in the pseudocode presented in Sec. 2 (and not randomly like in the mathematical model from Sec. 3). Than one container from each migration queue is allowed to transfer itself to another host. Each time step has a duration of the migration time T. Thus as opposed to the model from the previous section in takes into account finite network bandwidth.

First we performed 10000 simulations of situation corresponding to Eq. (7). The parameters were chosen in accordance with the presentation of Docker Swarm at DockerCon 2015 Conference: $n_0 = 49$, $N = 1000$ (at that conference a system of 1000 hosts with 50000 containers was demoed). The experiment ended when the numbers of the containers reached equilibrium $n_i = n_0$ and all the migration queues were empty $Q_i = 0$. In Fig. 1 we have a histogram of times needed to reach equilibrium t_0. We see that although 49 containers need to be transferred to reach the desired final state the average stabilization time is around 85 T. This is due to the fact, that the algorithm presented in Sec. 2 is probabilistic and sometimes more than one container chooses to migrate. Such containers interact with each other by entering the same migration queue.

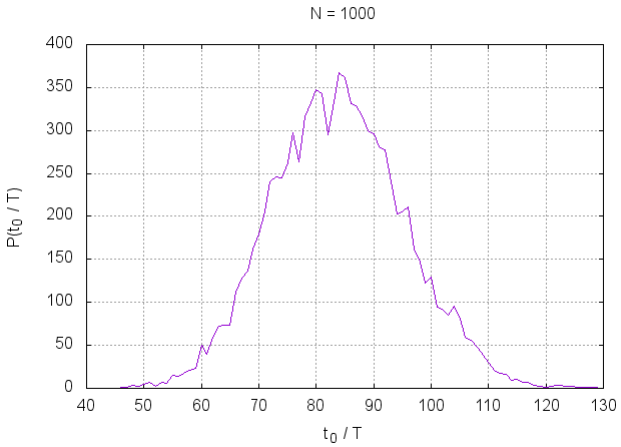

Fig. 1. Histogram of times needed to reach equilibrium t_0 for a system corresponding to Eq. (7).

Next the experiment was repeated for increasing number of hosts. The initial state was: one host in 50 empty and the remaining had 50 containers each. Therefore for $N = 50$ one host was initially empty, for $N = 100$ —two hosts, for $N = 150$ —three hosts, and for $N = 1000$ —fifty hosts were initially empty. The scaling behavior of the average value of t_0 is depicted in Fig. 2. We see, that the time complexity of the algorithm is of the order of O(N) in this case. Therefore although the mathematical model from Sec. 3 predicts time of the order of O(log N) to predict which container should migrate where, the majority of time in the case of Eq. (7) is spend on emptying the migration queues (this process is obviously of linear order).

Let us now present the same plots calculated for the second case corresponding to Eq. (7). They are presented in Figs. 3 and 4. Again sometimes more than one container chooses to migrate — this is the reason why the histogram from Fig. 3 is not centered around 49 T. Surprisingly the time complexity as depicted in Fig. 4 in this case is a logarithmic one. Thus the mathematical model from Sec. 3 in specific cases can very well describe reality.

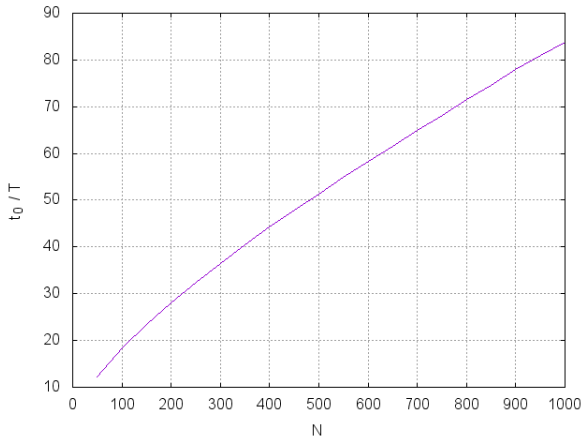

Fig. 2. Scaling of the average time needed to reach equilibrium t_0 with the number of hosts N. Initial conditions are given by Eq. (7).

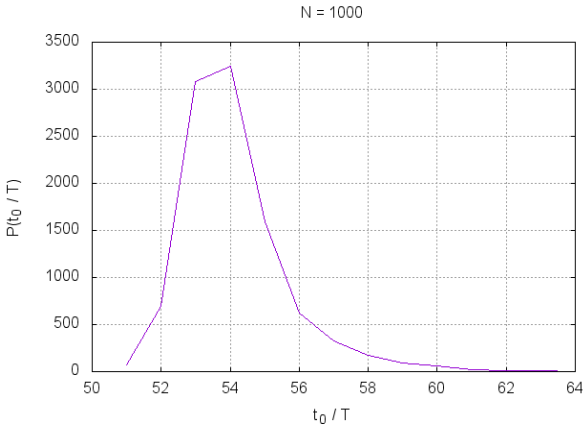

Fig. 3. Histogram of times needed to reach equilibrium t_0 for a system corresponding to Eq. (8).

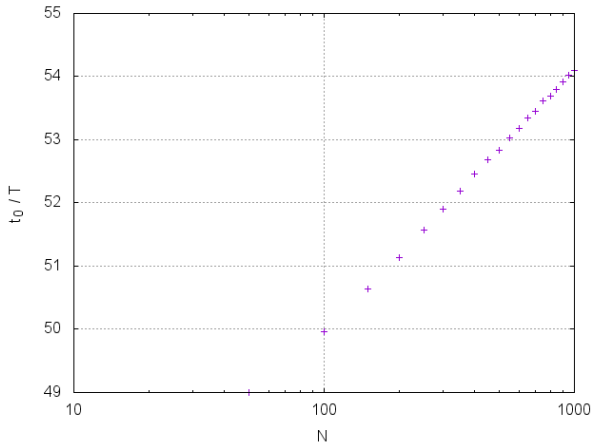

Fig. 4. Scaling of the average time needed to reach equilibrium t_0 with the number of hosts N. Initial conditions are given by Eq. (8).

5 Summary

In summary, an decentralized algorithm for load balancing of containerized microservices in the cloud was studied theoretically and numerically. Each container includes an additional mobile agent process which governs its migration to another nodes of the cloud. Thus the tasks running on the nodes of the cloud self-organize to maintain a constant load among the servers. The resulting system is resilient on server failures: a failing server can take down only a minimal number of tasks.

The performance of a similar centralized algorithm is of the order of O(N log N). First the numbers of containers on all hosts are sorted. Next the list is traversed from both sides and decision is made which containers should be migrated where. The decentralized algorithm studied in this paper gives the worst case time complexity of the order of O(N), and in specific cases even O(log N).

Swarm systems consisting of multiple autonomous agents exhibit complex behaviours. The interaction between the robots and its environment is very important to achieve the overall group performance. The analyzing of the system's behaviour using real and

simulated experiments is often expansive and time-consuming. Therefore, the observation of, for example, social insects can help in the extraction of ideas and models underlying natural systems and apply them to artificial problems. Using mathematical models we can efficiently study the swarm systems in order to better understand them as a whole.

In our paper, the problem of designing control policies that enable the swarm of agents (virtualization containers) to distribute themselves between multiple sites (servers) was solved using a system of differential equations that summarize the state transitions. However, one should not forget that in more sophisticated cases, interactions among the agents may lead to the transition probabilities that are a function of the number of agents in other states, and thus yield a system of differential equations that are time delayed and nonlinear. Hence, there are many directions for the future work. We would like to extend our linear model and implement nonlinear transition rules as well as time delays. The other idea could be to model large agent populations using partial differential equations. Also the use of stochastic differential equations should be considered.

References

1. J. Thönes, IEEE Softw., **32(1)**, 116 (2015)
2. E.A. Brewer, *Proceedings of the Sixth ACM Symposium on Cloud Computing*, 167 (2015)
3. B. Hindman, B., A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R.H. Katz, S. Shenker, I. Stoica, *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation*, 295–308 (2011)
4. J. Stubbs, W. Moreira, R. Dooley, *7th International Workshop on Science Gateways* (*IWSG*), 34–39 (2015)
5. C. Pahl, IEEE Cloud Comput., **2(3)**, 24–31 (2015)
6. M. Zhao, R.J. Figueiredo, *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, 5 (2007)
7. N. Kratzke, *CLOUD COMPUTING 2015: The Sixth International Conference on Cloud Computing, GRIDs, and Virtualization*, 165–169 (2015)
8. A. Mirkin, A. Kuznetsov, K. Kolyshkin, *Proceedings of Linux Symposium*, **2**, 85–90 (2008)
9. A.Y.U. Gital, A.S. Ismail, H. Chiroma, A. Abubakar, B.M.A. Abdulhamid, I.Z. Maitama, A. Zeki, *Information and Communication Technology for The Muslim World (ICT4 M 2014)*, 1–6 (2014)
10. M. Rusek, G. Dwornicki, A. Orłowski, *Advances in Systems Science, ICSS 2016. Advances in Intelligent Systems and Computing*, **539**, 142–152 (2017)
11. M. Rusek, G. Dwornicki, A. Orłowski, *Proceedings of 36th International Conference on Information Systems Architecture and Technology – ISAT 2015 – Part III. Advances in Intelligent Systems and Computing*, **431**, 75–85 (2016)
12. S. Berman, Á. Halász, M.A. Hsieh, *Bio-inspired Computing and Networking* (2011)