IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 12, 2020, accepted February 21, 2020, date of publication March 16, 2020, date of current version March 25, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2980852

Toward Bio-Inspired Auto-Scaling Algorithms: An Elasticity Approach for Container Orchestration Platforms

JOSÉ HERRERA AND GERMÁN MOLTÓ
Instituto de Instrumentación para Imagen Molecular (I3M), Centro mixto CSIC-Universitat Politècnica de València (UPV), 46022 Valencia, Spain
Corresponding author: José Herrera (jherrera@upv.es)

This work was supported by the Ministerio de Economía, Industria y Competitividad, Spanish Government, for the Project BigCLOE under Grant TIN2016-79951-R.

ABSTRACT The wide adoption of microservices architectures has introduced an unprecedented granularisation of computing that requires the coordinated execution of multiple containers with diverse lifetimes and with potentially different auto-scaling requirements. These applications are managed by means of container orchestration platforms and existing centralised approaches for auto-scaling face challenges when used for the timely adaptation of the elasticity required for the different application components. This paper studies the impact of integrating bio-inspired approaches for dynamic distributed auto-scaling on container orchestration platforms. With a focus on running self-managed containers, we compare alternative configuration options for the container life cycle. The performance of the proposed models is validated through simulations subjected to both synthetic and real-world workloads. Also, multiple scaling options are assessed with the purpose of identifying exceptional cases and improvement areas. Furthermore, a nontraditional metric for scaling measurement is introduced to substitute classic analytical approaches. We found out connections for two related worlds (biological systems and software container elasticity procedures) and we open a new research area in software containers that features potential self-guided container elasticity activities.

INDEX TERMS Auto-scaling, bio-inspired, sofware containers.

I. INTRODUCTION

The widespread adoption of Linux containers, and in particular Docker [1], as a mechanism for convenient application delivery has paved the way in the last years for the surge of the microservices architectural pattern [2] in which monolithic applications coded in a single programming language can be broken down into multiple polyglot services exposing interfaces. These are typically delivered and executed as containers managed by a Container Orchestration Platform (COP) such as Kubernetes [3] or Apache Mesos [4]. A COP acts as a scheduler for the execution of container-based workloads and provides secure access management to the pool of shared computing resources which are typically delivered in the shape of a cluster of computing nodes.

Previous literature agrees on the performance advantages of applications running on containers when compared to other virtualization technologies (see for example Felter *et al.* [5]). Indeed, an application running in a container can deliver a similar performance to when executed directly on bare metal. In addition, server density can be higher because no extra Operating System services are executed and, therefore, more containers can be executed per machine. However, microservices applications lead to faster creation, operation and removal of computing entities (containers) when compared to using Virtual Machines. This imposes a serious challenge for auto-scaling where more adaptable, precise and capable systems are required to manage the elasticity of large-scale fleets of containers belonging to multiple application architectures with dynamic elasticity requirements.

Adapting computational systems to the dynamic workload has been widely studied in previous works based on virtual machines and containers (Qu *et al.* [6], Hoenisch *et al.* [7]). Indeed, auto-scaling systems are already available for certain platforms, but the requirements of emerging application architectures requires a review of the methods used to perform auto-scaling in heterogeneous platforms of containers. In particular, it is important to understand the implications of

The associate editor coordinating the review of this manuscript and approving it for publication was Songwen Pei.

current research in other areas, for the field of elasticity of container-based computing platforms.

One of these areas is bio-inspired algorithms, where lifelike biological evolution is used as an inspiration source to solve a complex problem in computer science by finding an optimal solution, and a set of meta heuristics that mimic a biological process are adopted, typically to address optimisation issues [8]. Coding a system using an algorithm based on natural living sometimes introduces a convenient approach to solve the challenge [9]. Addressing auto-scaling of containers by means of bio-inspired algorithms may produce a viable option to quickly respond to changing requirements or load peaks.

To this aim, this paper introduces a set of novel bio-inspired algorithms aimed at supporting auto-scaling in container-based computing platforms. After the introduction, the remainder of the paper is structured as follows. First, section II provides an overall overview of the related state of the art. Next, section III underpins the similarities and relations between auto-scaling and bio-inspired algorithms. Then, section IV proposes two models for auto-scaling based on cell modeling. Later, section V incorporates a proposed model evaluation to assess options, parameters and performance. It also includes experiments using the best proposed model versus well-known algorithms using synthetic and real world workloads. Finally, section VI summarizes the main achievements of the paper and concludes the paper with future work.

II. RELATED WORK

This section summarizes the related work in the area of auto-scaling focusing on container orchestration platforms and bio-inspired algorithms.

A. AUTO-SCALING

A key feature of cloud computing is elasticity (Muñoz-Escoí and Bernabéu-Aubán [10]), where applications can dynamically adjust the computing and storage resources. On the one hand, vertical elasticity involves increasing or decreasing the amount of computing and memory resources of a single computing entity (typically a virtual machine in a cloud provider). On the other hand, horizontal elasticity requires changing the architecture of the application to run on a distributed fleet of computing nodes that can grow and shrink according to the values of certain metrics, such as the average CPU usage in the last t seconds across the available nodes.

If no human intervention is required to adjust those resources, then auto-scaling is taking place [11]. Well-known examples of auto-scaling services are ECS Service Auto Scaling [12], autoscaling groups of instances for Google Cloud [13] or Microsoft Azure Autoscale [14]. A rich taxonomy of auto-scaling systems is described in the work by Qu *et al.* [6] and in the work by Al-Dhuraibi *et al.* [15]. Dynamic vertical scaling has also been addressed in the past through the dynamic allocation of memory to Virtual Machines, as described in the work by Moltó *et al.* [16]. Even public Cloud providers are starting to provide this functionality to some extent, as is the case of Jelastic [17].

It is common to find centralised auto-scalers that have an overall view of the state of the platform and provide scaling methods, resource estimation or scaling timing. Other studies based on containerisation such as the proposal by Kukade and Kale [18] or the one proposed by Kan [19] also involve centralised systems. Autonomic resource provisioning has appeared recently as a rich research field. Some examples are explained in Calcavecchia *et al.* [20] proposing a complex architecture and requiring agents relations. Najjar *et al.* [21] proposes a double type of modules that run as agents while An *et al.* [22] focused on allocating network resources in dynamic environments. Chieu and Chan [23] describe a complex system with centralized data structures for coordination without any type of evaluation and Son and Sim [24] proposed agent-based testbeds focused on Server Level Agreements (SLAs) negotiation, price metric and time-slot utilities.

In an auto-scaling system, scaling decisions such as deploy or terminate a node in horizontal scaling are taken at a certain point of time and are typically followed by a *cool down* period in which no scaling actions are taken to avoid oscillations in the system. Therefore, scaling decisions are not continuous. In fact, the time between two consecutive scaling actions depends on the monitoring interval (slot time), and it is important to measure the optimal time distance between reconfiguration actions. On the one hand, a bigger distance between actions causes a worse adaptation of the system to the arising workloads. On the other hand, a lower time between scheduling actions may introduce an overload in the elasticity system. Some mixed methods are explored. For instance, Rad *et al.* [25] expose a vertical scaling procedure mixed with live container migration.

A new era of auto-scaling systems based on a wide variety of variables recently appeared. Examples of high-level metrics, such as request service time, appear in the work by Al-Dhuraibi *et al.* [26], whereas adjusting the estimation of service time is described in Kaur and I. Chana [27]. This suggests that the system must be capable of processing event-based workload prediction, provisioning using new pricing models, energy and carbon-aware systems (environmentally friendly) and more that will appear in the next years, considering previous variables too (classic economic cost and/or system performance). Although this work is focused on the reactive auto-scaling model [28], it is important to analyse that different nodes could use different values to evaluate options for auto-scaling.

A biological proposal for autoscaling is included in Moore *et al.* [29] where they evaluate a combination of time series prediction with an evolutionary algorithm using a centralized auto-scale architecture that start and stop virtual machines in a cloud computing platform. The reconfiguration interval or slot time elapsed 1 h according to most cloud infrastructure providers minimum period of charge, at the time when the study was carried out.

Virtual machine and software containers exhibit different values for resource provisioning, as demonstrated by previous works in the literature. The difference between provisioning a Container or a Virtual Machine has been assessed in Messias [30], in the work by Hussain [31] or in the work by Gupta *et al.* [32]. Indeed, containers are just processes executed by the underlying OS with some visibility restrictions and, therefore, do not involve the resource provisioning and boot up times required by VMs. Actually, it is very common to run containers (application delivery) on top of Virtual Machines (infrastructure provision).

B. BIO-INSPIRED MODELS AND CELL AUTOMATON

A cellular automaton is a mathematical model for a dynamic system composed of a set of cells that acquire different states or values. These states are altered from one moment to another in units of discrete time, i.e., that can be quantified with integer values at regular intervals. In this way this set of cells evolves according to a certain mathematical expression, which is sensitive to the states of neighbour cells, and is known as the local transition rule [33], [34]. Models based on individual agents allow personalised ''learn'', adaptation and reproduction [35].

Swarm intelligence is based on group behaviour of species like ants, bees, etc. These species have an intelligent behaviour without a centralized authority. Simple agents or bodies interacting individually with the environment or among them generate a global desired behaviour (see [36, Chapter 9]).

III. RELATING BIO-INSPIRED MODELS TO AUTO-SCALING

In this section, we analyse important underlying parallelisms and similarities between bio-inspired algorithms and auto-scaling actions. In both scenarios, decisions are taken at certain time intervals that affect the size of the population either by adding new individuals (i.e. deploying a compute node, which can be a container or a Virtual Machine) or killing individuals (i.e. shutting down a compute node).

The basic elements in a cellular automaton can be related to auto-scaling of containerised applications. A state set is possible both in cellular automata, and in auto-scaling systems. The initial configuration would be the same in both systems, and a transition function must be defined in both cases. Auto-scaling systems typically define a transition function for a centralised system. However, in cellular automata a transition function is defined for every single cell. By adopting the behaviour of cellular automata distributed auto-scaling can be performed without requiring a single centralised entity taking decisions about the scaling behaviour.

Some extrapolated advantages are: evolutionary algorithms are intrinsically parallel. They are a representation of an independent entity operating autonomously and, therefore, actions are taken individually. They work especially well for solving problems whose space of potential solutions is large and non-linear. They do well in problems with a complex adaptive landscape, such as those in which the fitness function

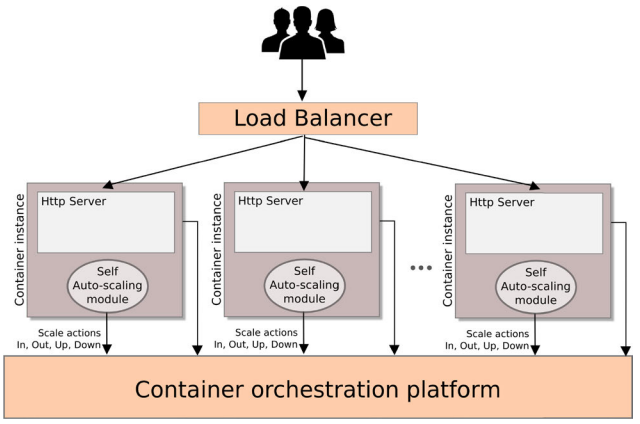


FIGURE 1. Logical distributed architecture proposal.

is discontinuous, noisy, changes over time, or it has many local optima [37]. They have the ability to manipulate many parameters simultaneously and exhibit a reduced time lapse for decision making. Also, decision in time t is faster and easier to take than in complex algorithms.

The main expected disadvantages are: Time taken for convergence; Configuration fine-tuning complexity; Mutation parameter definition; Fitness normalization or selection parameter configured sometimes by trial and error. Also, the algorithm may obtain incomprehensible solutions. The results could be out of scope, inefficient or incomprehensible from an engineering point of view. As a result of these disadvantages, the fitness function should be carefully designed and optimized.

The fitness function measures the differences between individuals and, therefore, decides which one is better. Some important characteristic are explained by Rebecca J. Parsons [38, Chapter 9], indicating that the fitness function is the way to discriminate between individuals and internal states, and it is desirable that similar fitness values are obtained for individuals with similar shared characteristics.

We do not aim to reach a fitness function where parameters are calculated and optimized with an evolutionary algorithm. Our proposal tries to distribute auto-scaling actions among existing containers to prove, in a preparatory work, that distributed auto-scaling represents a potential solution that mimics what happens in some living ecosystems.

IV. PROPOSED MODELS

For testing purposes, an application is a group of loosely coupled, fine-grained stateless services that communicate among them using a lightweight protocol. Requests are distributed into the application by means of a load balancer (Figure 1), which receives the user requests. Unless otherwise stated, experimental simulations use a round-robin load balancing algorithm, though this is changed for other distribution mechanisms to analyze the system behavior in other circumstances.

The initial proposed model has five possible environment adaptation actions on a classic scalable web architecture, as shown in Figure 2: horizontal scaling - *scale out* (denoted by *SO*), horizontal scaling - *scale in* (*SI*), vertical

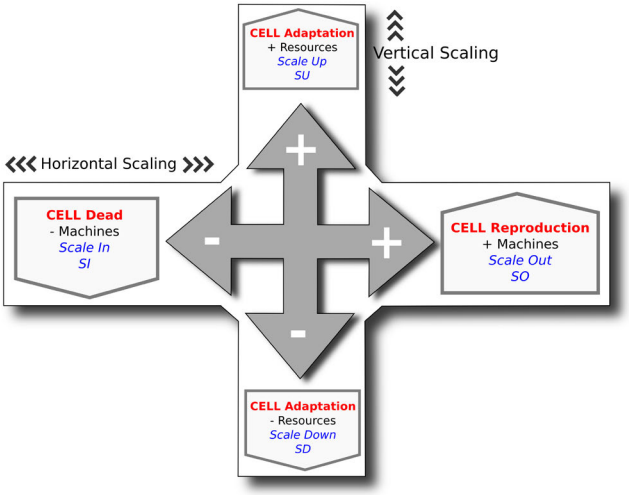


FIGURE 2. Relations for scaling actions, cell actions and provisioning actions.

scaling - *scale up* (*SU*), vertical scaling - *scale down* (*SD*) and no action (*N*).

Furthermore, probability functions are used in order to decide whether an action should be applied or not. This converts our system into a stochastic process. After a decision is selected (*SO*, *SI*, *SU* or *SD*), each cell performs the defined adaptation, for a time t, only if the probability is greater than a random value generated internally at the time for every defined action. Therefore, four probabilities are defined for four potential actions. *SIp* is the probability to become dead in horizontal scaling (*scale in*), *SDp* is the probability of adaptation in the case of vertical scaling (*scale down*), *SUp* probability for vertical scaling, but in *scale up* and *SOp* is the probability to perform an action of horizontal scaling (*scale out*), as described in Figure 2. Notice that the probability of these actions prevents an action to be concurrently taken by all the containers, as it would happen, for example, during a spiky workload increase. Therefore, it can be interpreted as the ability to adapt the system for every action. See Algorithm 1 for a cell algorithm for auto-scaling decision.

A. NORMALIZED EXTENDED RESOURCE METRICS (NOX)

Normalisation is required in order to compare different metric values, which are referenced as X_t in time instant t. This NOX (NOrmalized eXtended resource metric) is created based on the original metric values as described in equation 1.

$$X_t = a_0 * f_0(P_{0_t}) + a_1 * f_1(P_{1_t}) + \ldots + a_n * f_n(P_{n_t}) \quad (1)$$

where $a_x >= 0$ indicates the participation of every function (f_n) in NOX, $a_0 + a_1 + \ldots + a_n = 1$, P_{x_t} represents the parameters for sub-functions in time instant t and function $f_x <= 100$, $x \in [0..n]$ (n represents the total number of components in the equation). It takes values from 0 to 100 allowing us to compare systems in the way we desire. This sub-function represents a part of the NOX function with different treatments according to the variables defined (e.g. $f_1 = \%CPUload, f_2 = \%RAMuse, f_3 = IO\ operations * 100\ /\ max\ IO\ operations$, etc.).

Algorithm 1 Cell Algorithm for Every Decision in Auto-Scaling Module

```
while true do
    Wait (T)
    p=GetRandomNumber(0,1)
    NOX=GetNOXValue()
    if (NOXbetween(SO_L, 100)) ∧ (p >= SOp) then
Scale-out
        Action
    else if (NOXbetween(SU_L, SO_L)) ∧ (p >= SUp) then
Scale-up
        Action
    else if (NOXbetween(SI_L, SD_L)) ∧ (p >= SDp) then
Scale-down
        Action
    else if (NOXbetween(0, SI_L)) ∧ (p >= SIp) then
Scale-in
        Action
    else
        No Action
    end if
end while
```

The desirable features for NOX values are described by Khurana [39] for software metrics, among others: Consistent and objective, easy to calibrate, easy to obtain and robust.

B. AUTO-SCALING SELF-SUFFICIENT CELL MODEL (SCM)

A first analysis is made trying to maintain cells as independent as possible. The main feature of this model is the lack of direct interaction among the cells. The two main characteristics of the SCM algorithm are the NOX function applied (see section IV-A) and the transition function used. This initial model is interesting because if we define a NOX function that uses only local parameters, the model would reduce container data interchange and network traffic.

The SCM model has multiple options relaying on the NOX function, action limits or data from the previous status. Some initial alternatives are explained as follows:

- SCM-A option. The transition function is composed of a NOX value that only considers %CPU used ($X_t = \%CPU$) and a simple transition function (see Table 1) where X_t represents the NOX function. In this table every line represents a range of response and possible action depending on the NOX value. The column "Likely cell reaction" represents the name in a bio-inspired scenario whereas the "Auto-scaling name" column is the action name in an auto-scaling scenario. The last column ("Action") is a representation of the activity carried out.
  After the NOX evaluation and the transition function, an action is obtained (*SO*, *SI*, *N*, *SU* or *SD*), and the probabilistic correction explained at the beginning of section IV is carried out to prevent that every cell performs the same action. In Table 1, the variables SI_L, SD_L,

TABLE 1. Transition function definition for SCM-A option.

Function	Likely cell reaction	Auto-scaling name	Action
$SO_L <= X_t <= 100$	Reproduction	Horizontal scaling (*scale out*)	SO
$SU_L <= X_t < SO_L$	Adaptation	Vertical scaling (*scale up*)	SU
$SD_L <= X_t < SU_L$	Not action	–	N
$SI_L <= X_t < SD_L$	Adaptation	Vertical scaling (*scale down*)	SD
$0 <= X_t < SI_L$	Dead	Horizontal scaling (*scale in*)	SI

TABLE 2. Transition function definition for the SCM-B option.

Function	Likely cell reaction	Auto-scaling name	Action
$SO_L <= X_t <= 100$	Reproduction	Horizontal scaling (*scale out*)	SO
$(SI_L <= X_t < SO_L)$ $and(X_{t-1} <= X_t)$	Adaptation	Vertical scaling (*scale up*)	SU
$(SI_L <= X_t < SO_L)$ $and(X_{t-1} > X_t)$	Adaptation	Vertical scaling (*scale down*)	SD
$0 <= X_t < SI_L$	Dead	Horizontal scaling (*scale in*)	SI

SU_L, SO_L represent the thresholds defined in the model in order to take actions. SI_L is the upper limit of dead range action. Values of X_t between SI_L and SD_L cause *scale down* actions and SU_L is the lower limit for *scale up* whereas SO_L is the upper limit. The upper limit is defined by SO_L that is the lower threshold for *scale out* actions.

- SCM-B option. In order to simplify the possible actions applied to the managed system, the "Not action" (N) is removed. This case is an adaptation of SCM-A, where the range *Not Action* disappears and vertical scaling actions are joined to create alternative options using the NOX value from the previous iteration (X_{t-1}). The transition function is defined in Table 2. Variables SO_L and SI_L maintain the same meaning and vertical actions are made according to the previous NOX function. Therefore, we are prioritizing horizontal scaling and keeping limited the vertical scaling actions.
- SCM-C option. The transition function is the one defined for SCM-B in Table 2 and the NOX function is $X_t = 100 * \frac{Qsize}{Qlimit}$. In scenarios involving a web application where requests are demanded not periodically (with an unpredictable not periodic pattern in time), Q_{size} is the number of pending FLOPS and Q_{limit} is the value of the maximum number of pending requests.

More complex NOX and transition function can be defined depending on the system response that we need, but these cases explain the typical behaviour of an auto-scaling system with well-known system parameters.

For the sake of clarity and simplicity, all the references to SCM will correspond to the SCM-B option.

C. AUTO-SCALING INTERACTIVE CELL MODEL (ICM)

Auto-scaling Interactive Cell Model is an example of a system where cells/containers know information about adjacent cells/containers directly. This requires data interchange among cells, directly or through a mediator service.

Action probabilities and transition functions are similar to the ones described in the SCM models. However, the NOX function definition is different from ICM models. We need a function that joins information about nearby cells and local information. The set of cells closer to a given one is denoted as S_t^n.

$$X_t = \alpha * X_t^{local} + \beta * X_t^{neighbour} \tag{2}$$

$$X_t^{neighbour} = a_0 * f_0(S_t^0) + a_1 * f_1(S_t^1) + \ldots + a_n * f_n(S_t^n) \tag{3}$$

where $a_x >= 0$ is the participation of every sub-function in the neighbour part of the NOX function, $a_0 + a_1 + \ldots + a_n = 1$, $\alpha + \beta = 1$, $\alpha >= 0$ and $\beta >= 0$, $f_x <= 100$, $x \in [0..n]$ and S_t^n represents a set of cells defined at time instant t. When $\beta = 0$ this equals to the SCM model. X_t^{local} is equal to the SCM model in equation 1. The transition function is the same used for the SCM-B option (Table 2) and the probabilities defined in the interval maintain the same interpretation. The neighbour function $X^{neighbour}$ is based on a linear representation of cell space. This is built with only one dimension, the easiest representation for a n-dimensional space to render cells.

V. MODELS EVALUATION

After explaining our proposal, we run several simulations to evaluate the accuracy of the models. Tests were performed under controlled conditions explained in the following paragraphs.

In this case, we rely on simulation techniques to show the best performance cases and to avoid error configurations that would render the model unusable. Running these models in a production infrastructure (using real workload, actual hardware and real total number of elements) would involve a large-scale number of resources that are not justified for a first problem approach. In a stochastic system, such as this one, we run simulations several times in order to validate output values with multiple combinations of input parameters. This technique requires a lot of computational power to be executed a relevant number of times, thus discouraging the use of a real production infrastructure.

The first feature observed is that the transition function is related to the NOX function. In our case, the NOX function reviews node execution values and the transition function changes the total number of nodes (horizontal scaling) or node execution parameters (vertical scaling). Relationship between NOX function and transition function must be well designed to prevent wrong behaviours. A well-known case is an incorrect horizontal scaling action in cases of network bottlenecks.

All models were run using a self-implemented code accessible in GitHub [40]. This simulator emulates every node as a class instance that adjusts its parameters using the previously defined transition function.

The following sections validate the models through simulations and verify how the system responds to load changes with the proposed algorithms. After that, we define the process

variables that will be used to compare the proposed models. Finally, relationships among variables and the results are confirmed or belied using graphical methods.

A. GLOBAL CONDITIONS

The general conditions for testing are described in the following paragraphs together with the methods used for data collecting, monitoring and analyzing.

1) MEASURING ELASTICITY

Generally, auto-scaling is done to optimize the system in terms of cost and/or performance like the response time or throughput. Indeed, in the work by Ai *et al.* [41] the authors propose an extended list of values that can be used to evaluate scalability proposals. Workloads in a production system are highly unpredictable and do not typically exhibit clear patterns [42]. Furthermore, without regarding prediction systems, all scalability systems have a non-defined time delay between action and response. Therefore, we needed some strategy to adjust a correct evaluation in addition to a correct response.

We have four statistical metrics for evaluation:

1) **MAE** (Mean Absolute Error). It is a measure of difference between two continuous variables.

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|e_t| \tag{4}$$

2) **RMSE** (Root Mean Squared Error). Larger errors have a larger effect on this value. Thus, it is very sensitive to outliers.

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}e_t^2} \tag{5}$$

3) **EVS** (Explained Variance Score). The best possible value is 1. Lower values are worse.

$$EVS = 1 - \frac{\sum_{t=1}^{n}(e_t - \bar{e})^2}{\sum_{t=1}^{n}(y_t - \bar{y}))^2} \tag{6}$$

4) **R2S** (Coefficient of Determination or R-Squared value). The best possible value is 1 and it can be negative (because the model can be worse).

$$R2S = 1 - \frac{\sum_{t=1}^{n}e_t^2}{\sum_{t=1}^{n}(y_t - \bar{y})^2} \tag{7}$$

where: $e_t = V_{real,t} - V_{expected,t}$, $y_t = V_{real,t}$ and $\bar{y} = mean(y_{series})$

Alternatively, Dynamic Time Warping (DTW) is a time series alignment algorithm developed originally for speech recognition by Sakoe and Chiba [43]. It aims at aligning two sequences of feature vectors by warping the time axis iteratively until an optimal match between the two sequences are found. According to our purposes, the best match between two sequences is obtained by measuring the DTW distance. It is proposed as an alternative to e_t calculus for MAE (Equation. 4).

In order to compare two time series, DTW distance perfectly fits our purpose. The first time series represents the resources needs in a t time in comparison with a second real-time series equal to our available resources. Then, the DTW distance between the two series is a measure of the adaptive capacity for the system designed (available resources) and system desired (resource needs). When it is near to zero, it is a proper value. Therefore, the greater the value the worst it is considered. However, the DTW algorithm only provides modulus ($|x|$) values comparative (positive distances) for best fitting points. An additional problem is that using modulus values no difference exists when resource load values are greater or lower than the reference series. This implies no evaluation of under-provisioned or over-provisioned behavior, like MAE or RMSE do.

When the load of total requests exceeds the system capacity, the served requests must wait until there is processing time available. The DTW algorithm does not distinguish between over-provisioned and under-provisioned scenarios. To model overload, we introduce a store for requests that are partially processed. The size of this store indicates resource needs to finish the jobs in container (e.g., if 10 MFLOPS of CPU resources are pending, the store size will be 10). To prevent under-provisioned system times from not being analyzed in our study, we included a new result value for all simulations called queue size. In this case, the total queue size (Q_{size}) is the value selected for system evaluation. This experiment was designed for testing purposes, validating that over-provisioned systems have $Q_{size} = 0$ values for all the executions and under-provisioned show $Q_{size} > 0$.

2) ALGORITHMS FOR TESTING AND COMPARATION

Together with our bio-inspired algorithm proposal, we used two well-known centralized algorithms too:

- *Common*. This simple algorithm allows adapting capacity depending on $CPU_{utilization}$ like Amazon's Simple and Step Policies EC2 Auto Scaling feature (SS) [44]. It is simple and easy to code, but beneficial and efficient for simple applications. When $CPU_{utilization}$ is under 30%, we reduce the total number of containers running; over 70% we create a new container.
- *Prediction*. This algorithm is similar to the previous one, but we tried to estimate future values using SARIMA(5,1,0), also known as seasonal ARIMA, implemented in Python's *statsmodels* library to forecast the CPU usage in the next time slot instead of using the actual $CPU_{utilization}$. SARIMA prediction uses all series of data available in order to estimate the next value. After this calculus, we use similar limits to those used for the Common algorithm introducing proportionality to container creation. If the load predicted by the algorithm is not satisfied by adding only one node then it will create the necessary additional nodes.

3) SERVERS LOAD FOR MEASURING

We analyze our proposal using three time series. The first load used is a synthetic proposal -SYNTLoad- (see Figure 3),

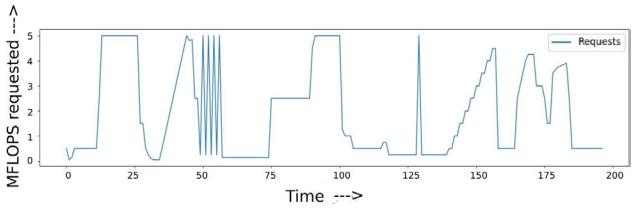


FIGURE 3. Synthetic load series (SYNTLoad).

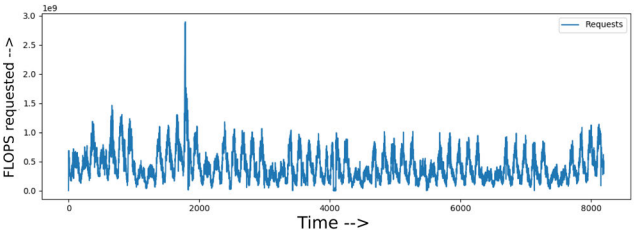


FIGURE 4. 1995 NASA series (NASA95).

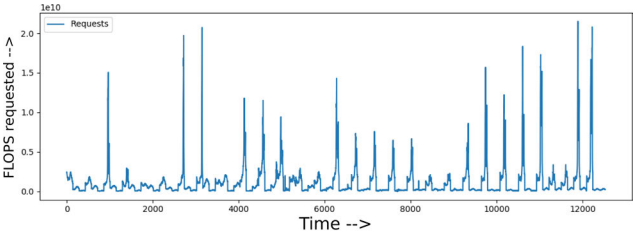


FIGURE 5. 1998 FIFA World Cup series (FIFA98).

where different web-based workload patterns across time are shown. Based on the work by McNaughton [45] and Gill *et al.* [46], we take several workload patterns for testing purposes. This creates a non-cyclic and non-seasonal time series that will serve us to better evaluate the models and configurations with unpredictable workloads. The second one is a log from August 04 to 31, 1995 of HTTP requests to the NASA Kennedy Space Center WWW server in Florida -NASA95- (see Figure 4) [47]. Finally, we have a log from France FIFA World Cup Series 1998 -FIFA98- (see Figure 5) [48]. With the aim of reducing real load extension from FIFA98 and NASA95, the total lines are grouped in a single request for each 10 minutes. Therefore, the number of requests is the same, but they are grouped in longer time periods.

To assess the models, in the simulator, load requests are transformed as resource load using configuration variables. All executions carried out were configured to use FLOPS as the basic measurement unit for output values, summaries and visualization data for the sake of better understanding.

B. SELECTING BEST PROPOSED ALGORITHM

So far, we have presented models and the way to measure and compare them. Firstly, to validate our proposal, we evaluate models with SYNTLoad in this section. After that, in section V-C, we will compare the best proposed model with real loads (FIFA98 and NASA95) and well-known auto-scaling algorithms (common and prediction).

Previously to experiment validation, we tried to relate parameters provided on simulation and outputs (see Figure 6).

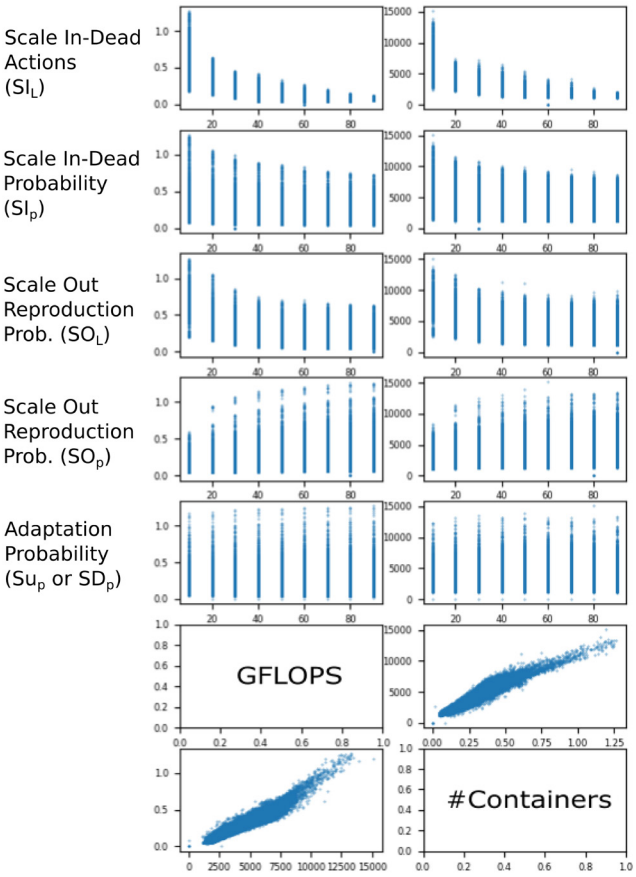


FIGURE 6. First multivariable analysis for SCM simulation.

In this case, varying five input simulation parameters (SI_L, SI_p, SO_L, SO_p and SD_p or SU_p) the range of parameters are $SI_L \in [0..100]$, $SI_p \in [0..1]$, $SO_L \in]SI_L..100]$, $SO_p \in [0..1]$, SD_p or $SU_p \in [0..1]$ (see Table 5 compared with Table 4). We produced two outputs in a controlled environment, aggregate DTW distance and #containers, both shown as columns. All other possible parameters in the simulation were maintained with constant values. As we use percentage variables (SI_p, SO_p and SD_p or SU_p) to obtain a result, we run ten times the same simulation and the result was the average of all the executions. Rows are identified as limits (SI_L and SO_L) or probabilities (SI_p, SO_p and SD_p or SU_p). Limits use steps of 10 whereas probabilities have a step of 0.1 and both are represented in the X-axis and normalized in ranges from 1 to 100. Y-axis is normalized too. By varying parameter values in the X-axis, we created all the combinations for DTW distance and *#containers* (represented in the Y-axis). The relation between DTW distance and #containers can be obtained as the result of previous variations and is represented in the lower part of the multivariate analysis.

Also, we generate Figure 7 with a multivariate analysis of independent interval variables *MinCellsRunning*, *MinVerticalLimit* and *MaxVerticalLimit* related to the dependent variables DTW distance difference (measured in GFLOPS) and total number of containers (#containers) produced in the simulation.

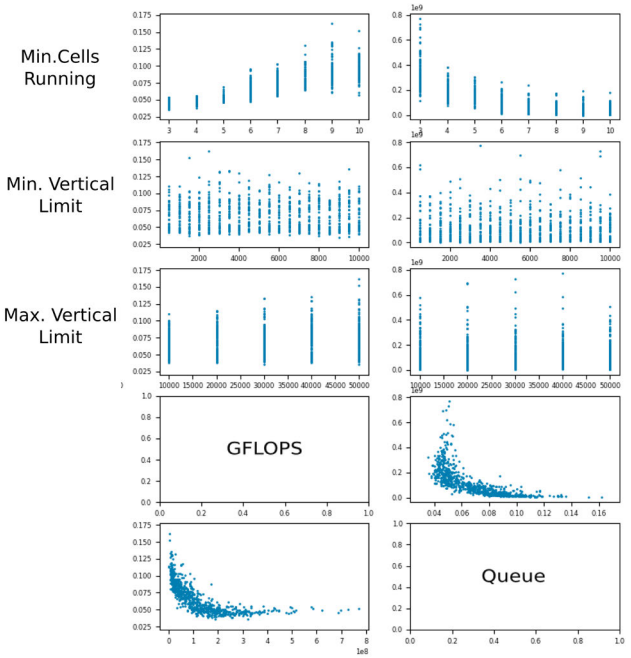


FIGURE 7. Second Multivariable analysis for SCM simulation.

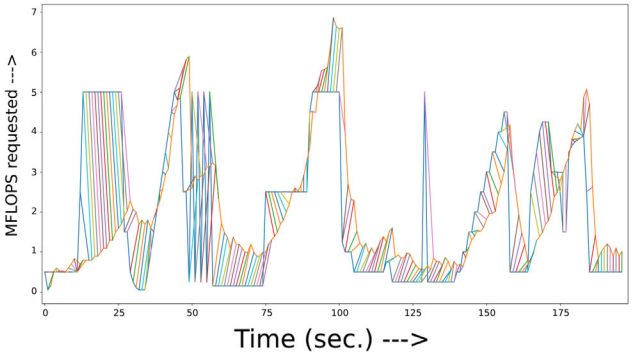


FIGURE 8. DTW distance results for SCM-B simulation.

Multivariate analysis involves observation of more than one outcome variable at a time in order to reduce a large number of variables to a smaller number of factors. The relation between variables and results (aggregate DTW distance and #containers) are not clear in most cases. Only SI_L, SI_p and *MinCellsRunning* variables have an influence on the results (dependent variables). It looks a good variable related to Scale-in actions (SI). Only with a low value of SI_L a good auto-scaling response is achieved with an appropriate number of containers. The best value of *MinCellsRunning* is around 9.

1) MEASURING SCM EXPERIMENTS

This section validates the SCM model for auto-scaling systems in order to gain better understanding about its behavior under SYNTLoad situations (Figure 3). The dynamic container provisioning output for executions is shown in Figure 8. In the same way, for all experiments, we capture the DTW distance (total distance lines between the load applied and the system capacity obtained) as the main approach to compare experiments.

TABLE 3. Compilation of featured experiment results.

Experiment	Tot. DTW distance	Tot. processing containers	Ratio Q_{size} / #containers	Total Q_{size}
SCM1	194	4,986	1.492	7,440
SCM2	57	1,125	0.005	6
SCM3	418	7,493	0.021	158
ICM1	519	11,884	0.072	864

TABLE 4. ICM1 experiment limits and values. Including variables options.

Action code	SI			SD or SU			SO	
NOX limits	0	<—>	SI_L	<—>		SO_L	<—>	100
Act.percentage		SI_p		SD_p or SU_p			SO_p	

TABLE 5. SCM1 experiment limits and values.

Action code	SI			SD or SU			SO	
NOX limits	0	<—>	30	<—>		90	<—>	100
Act.percentage		70%		50%			10%	

The **SCM1 experiment** runs the SCM-B model with constant variables. Steady values are presented in Table 5. The first line shows three actions that can be performed (*SI - scale in* / *SD - scale down* or *SU - scale up* / *SO - scale out*). The second line displays function limits (0, $SI_L = 30$, $SO_L = 90$, 100) and the last row includes final action percentage (*SIp* - Dead Probability (70%), $SUp = SDp$ (50%), *SOp* - probability of *scale out* (10%)). The limit between the *SD* and *SU* actions is explained in Table 2. Reading Table 5 by columns: *SI* action (*scale in*) is triggered when NOX value is from 0 to 30 with a probability of 70%. *SU* or *SD* action is triggered when NOX value is between values 30 and 90, with a probability of 50%. Finally, the *SO* action (*scale out*) is triggered with values of NOX between 90 and 100 and only 10% of the times. These values are selected initially as a starting point for the system in a heuristic way. The following experiments adapt these values. The decision for vertical scaling is made based on previous resource load. If the preceding load is less that the current resource load requested then the vertical decision aims at increasing the capacity (SU). Conversely, the decision is to decrease (SD) the resources allocated to the cell. Figure 3 (SYNTLoad) shows the total load demanded for total requests in our experiment. See Table 3 for detailed results of SCM1 experiment.

In the **SCM2 experiment**, the load balancing algorithm is modified. Preceding experiments involved simulations using a round-robin load balancing algorithm. In this experiment, an agent based algorithm for load balancing is integrated in the simulator. It distributes requests based on the container load. With this approach, even if we assume the same load in every request, not all container will receive the same number of requests. Instead, we use the processing capacity for all nodes as a heuristic to distribute the requests among the least loaded cells. Using only horizontal scaling, no capacity differences among nodes exists. All the nodes feature the same processing capacity. To enable nodes capacity variation, we need vertical scaling that does not exist if the dead superior limit variable (SI_L) and horizontal lower limit variable (SO_L)

are the same (center range is reduced to zero). See Table 3 for detailed results of SCM2 experiment.

SCM3 experiment. For this experiment, we created a system with improved characteristics in vertical scaling. Indeed, scaling up/down (vertical scaling) is sometimes more efficient than scaling in/out (horizontal scaling) (see Rad *et al.* [25]). Indeed, adding a new node requires more time than allocating further CPU, memory or other resources to an existing one. Therefore, we design the experiment favouring additional vertical scaling actions compared to the previous experiment. The parameters used in this experiment will reduce the range for *SI* and *SO* as much as possible, setting SI_L to 5 and SO_L to 95. This creates a wider range for vertical scaling (from 95 to 5). The request time series is the same one used in all the previous experiments: SYNTLoad.

2) MEASURING ICM EXPERIMENTS

This section introduces the ICM model under basic setup conditions.

ICM1 experiment tries to improve the results obtained in the SCM experiments and apply the ICM model explained in section IV-C with new external data sources. ICM cells collaborate directly with other cells to improve total throughput and reduce total Q_{size}. In this case, we use a straight line (one dimension) living space where cells remain until death. Every cell has only cells to the left side and to the right side because the space is continuously configured where the last cell has the first cell as neighbour and vice versa. When a cell looks for cells adjacently it is possible to see a distance greater than 1. The parameter *icm_size* indicates the total cells seen on both sides used for the computations. Other options, such as a two-dimensional space or a multidimensional mesh are possible, but they are not addressed in this document because the resulting analysis is more complex.This experiment brings in a major problem when we consider intercommunication between neighbour containers. The use of a simulation environment allows us to evaluate models independently from the communication strategy among the cells, which can be achieved by a message bus or a service mesh.

We focus on whether horizontal scaling is enough to adjust to workloads and compare the results against other experiments. Therefore, in this model, no vertical scaling (up or down) is produced, all adaptation is made using horizontal scaling (using in/out scaling), *SI* or *SO* decisions. The NOX formula is similar to the one used for SCM-B but now the value is a combination of local data and neighbour's information. All information about local resources used can be mixed with all average information from then right side and then left side cells using a weights formula (see equation 2 parameters X_t^{local} and $X_t^{neighbour}$). Value for %Local is $\alpha = 15$, this means a %neighborhood value for $\beta = 85$.

For testing purposes, all the information from neighbours is obtained from a centralised database periodically populated with the information from the neighbours themselves. See Table 3 for detailed results of the ICM1 experiment.

TABLE 6. Summary table comparing Bio algorithms with vertical scaling (SCM2) and without vertical scaling (SCM3).

Load Name	FIFA98		NASA95	
Vertical Auto-scaling	With	Without	With	Without
Total Distance DTW	11,165,275	10,782,642	15,425,883	2,759,028
Total Q_{size}	4,603	16,505	1,598,359	2,188
Total Container Used	446,390	2,203,148	3,357,354	576,863

3) RESULTS ANALYSIS

The results in Table 3 highlight important details comparing the four data experiments. According to the Tot. DTW distance column, the best algorithm is SCM2, where the result obtained is the closest to 0. The experiment that uses the lowest number of containers to run the workload is SCM2 as well, using 1,125 containers across all the experiment. Considering the Q_{size} column, the lowest size is achieved in experiment SCM2. Therefore, SCM2 stands out as the best approach in the model proposed. ICM1 uses a large number of containers (11,884), a 58% more than SCM3 (7,493). SCM1 was significantly worse at Q_{size}. The ratio $Q_{size}/\#containers$, highlights a similar situation: the best ratio is obtained for SCM2, whereas SCM3 and ICM1 show similar response times and the worst is SCM1. Therefore, mixing a bio-inspired approach with a local balancing strategy stands out as the best approach.

C. COMPARING WITH REAL WORLD LOADS

This subsection describes the results obtained as a combination among loads (explained in previous section V-A.3) and algorithms (two exposed in section V-A.2 and models proposal of section IV-B). Trying to explain the most important data in Table 7 by columns.

SYNTLoad: Notice that, in Table 7, the Total Container Used in Bio (running SCM2 option) is higher than the one used in other real world algorithms used (Common or Prediction) but Total Q_{size} is lower. This offers an explanation of why a lower Q_{size} is obtained for this proposal. Using DTW, instead of evaluating $error = V_{expected} - V_{actual}$ to calculate MAE, we get similar values for three proposals.

FIFA98 Load: Bio algorithm was run using a combination of Vertical and Horizontal Scaling (see V-C.1). The values shown for other variables are similar to the Bio algorithm. MAE DTW, MAE and RMSE are similar too, but EVS and R2S not. The value of Q_{size}, related to response time, is better for our Bio algorithm. Predictive worst results are justified for a poor forecast and not performing optimal seasonal adjustment of time series analysis with outliers. Indeed, *Common* is a very simple algorithm to adjust to a complex workload.

NASA95Load: Bio algorithm was run using only Horizontal Scaling (see V-C.1). This load shows differences with respect to other two loads. It is true that Bio uses twice as many containers, but Q_{size} is very low compared with other algorithms. MAE DTW has the worst values for Bio too. The Predictive algorithm is better in this case, but Common also obtains a good result.

TABLE 7. Summary table comparing loads and auto-scaling algorithms results.

Load Name	SYNTLoad			FIFA98			NASA95		
Auto-scaling algorithm	Common	Bio	Predic.	Common	Bio c [c]	Predic.	Common	Bio	Predic.
Auto-scaling # points		197			12,528			8,199	
Total Distance DTW	5,710	7,006	5,778	11,144,499	11,165,275	11,121,795	1,380,128	2,759,028	1,412,102
Total Container Used	5,767	17,596	5,780	1,104,654	466,390	1,108,445	295,180	576,863	291,925
Total Q_{size}	196	5	193	3,095,636	4,603	2,956,310	750,523	2,188	823,881
MAE DTW a [a]	29.0	35.6	29.3	889.6	891,2	887.8	168.3	336.5	172.2
MAE a [a]	31.4	42,2	31,6	908.5	911.5	906.5	174.1	381.6	177.2
RMSE a [a]	37,2	61,4	37,2	1,792.6	2,001.1	1,789.9	228.9	588.6	232.6
EVS	0.048	-0.872	0.029	0.140	0.631	0.142	0.282	-2.952	0.254
R2S	-0.039	-1.825	-0.035	0.129	-0.084	0.132	0.225	-4.123	0.199
Capacity Coef.b [b]	2.44	0.35	2.33	6,83	0.66	6.81	3.40	0.32	3,45

[a] x1.000.000, [b] Coefficient for Maximum Requests/Maximun Processing Capacity , [c] Using Vertical and Horizontal Auto-scaling

1) CONFIRM HORIZONTAL OR VERTICAL SCALING BEST RESULTS IN REAL WORLD

In previous sections and Table 3, we compared models using a SYNTLoad. To confirm the results, we run the best modes (SCM2 and SCM3) with real loads. This generates Table 6 where the most important values are presented. For FIFA98, the best results correspond to vertical-horizontal mix scaling option (SCM2) but, for NASA95, the best results are without vertical scaling (only horizontal) using SCM3. We estimate that this is due to the load peaks in FIFA98 that are not present in NASA95. Our best performance hypothesis for Vertical Scaling, tested on SCM3, is not confirmed and performance is clearly subordinate to load type.

2) LOAD/ALGORITHM COMBINATION OUTCOMES

The best results for our proposed model are accomplished in loads with unpredictable series points called outliers. Most predictable loads are better for other two algorithms. DTW used, as error measure, is useful to compare the shapes of series in unpredictable cases. Our Bio algorithms are always over-provisioned, and they have better measures for response times that need be validated in further research.

By putting response time first, using a Bio algorithm is the best option. Contrasting loads, if it is foreseeable, a Prediction algorithm or a Common algorithm are better selections, but if we have a non-cyclic and non-seasonal time series load, Bio options would be a possible selection as a result of a better system performance adjustment using an over-dimensioned number of containers but it clearly increases the cost of the solution.

3) FEASIBILITY OF SCALE ACTIONS

If we want an agile response to change, we need to estimate the workload according to previous data or design a fast response time system (with Q_{size} low). As a final assessment of our Bio proposal, we need to analyze the economics of scale. On Table 8, we reference not only the containers used, from Table 7, but also variations and peaks in the simulation.

Adding, removing or modifying container characteristics is a task that we need to measure when using a high number of containers in our proposal. We need an appraisal of the cost of running containers taking into account that the Bio algorithm increases the total adaptation actions made as well.

TABLE 8. Scale cost for Bio algorithm.

Experiment	SYNTLoad Bio	FIFA98 Bio	NASA95 Bio
Auto-scaling points	197	12,528	8,199
Total container used	17,596	466,390	576,863
Mean containers	89.31	37.22	70.35
Container peak	351	2,911	910
Max # scale-out/in actions	92/280	1,354/1,379	440/748
Mean container lifetime	48	1,566	132

Vertical scaling is an elementary response action for a temporary lack of resources, adding more CPU capacity or memory to our under-provisioned node. Being a very limited action in time, this has not an effective cost for computing platforms, but vertical scalability has a limit and the cost gradually increases (see Henderson [49], Openshift and [50]). In container platforms with pay-per-use model some vertical resizing is made using a stop-start model with no possibility of live vertical scaling. For example, Amazon does not support live vertical scaling of EC2 instances. In contrast, we have Jelastic [17] that define a vertical scaling like our proposal based in a scaling limit of resources in two parts, reserved and dynamic.

The cost of horizontal scaling includes the total number of containers running in a time period and our solution sometimes doubles the number of containers with respect to other strategies. But bio-inspired reduces the total time that a container is up. This means that it creates a large number of short-lived containers. Using a Bio algorithm with loads based on peaks, we ensure right scaling and controlled cost.

Another important issue is container creation times. In [51], authors create 1 million containers in 266.7 s. This is a rate of nearly 3,750 containers per second. Greater container scale-out demand is 1,354 containers for a slot time (similar for scale-in). Focusin on Container Orchestration Platforms there was no noticeable difference in the launch time using 30,000 containers in Docker Swarm or Kubernetes (see [52]–[54]).

Sharper load peaks or series with greater number of outliers are solved better with horizontal scaling and repetitive load are conveniently solved with vertical scaling. To put the two possibilities together, a possible solution is to equip our models with a peak detection in request series.

VI. CONCLUSION

An early study was made to handle a bio-inspired distributed algorithm in a container orchestration platform by employing a simulation software, introducing two experimental models that help to distribute decision making across the involved nodes.

Achieving low Q_{size} values, whereas exhibiting a high processing capacity (low DTW distance) are opposite goals. In some cases, attempting to improve a metric by configuring certain parameters results in worsening other metrics. A compromise solution is the most appropriate solution. Oscillation problems of auto-scaling algorithms are limited in bio-inspired models and appear to be controlled by configuration parameters. Suitable models mix vertical and horizontal scaling, joining all actions in an easy way and allowing other potential scaling actions. NOX functions are defined as a tool to engage reactive algorithms or self-sufficient with cell-related models. An algorithm, designed initially for time series alignment in speech recognition (DTW), is used to compare multiple options. We perceive it as a good alternative to equate complex time series produced in auto-scaling actions (very long sets or very detailed sequences).

Bio-inspired models do not offer an upper bound for the number of containers running at a t time. Scale up operations (SU) can be done indefinitely until the total capacity of the hardware is exceeded. Furthermore, scale out (SO) operations can be unlimitedly performed for all cells. Dead prevention for an inferior limit of total containers (to avoid selection of SI action for all cells) is complex too. The SCM model reveals features of a very sensitive system to multiple parameters. A proper configuration is a multiple parameter combination where it is not easy to calibrate dynamic response to multiple situations. The correct configuration of a multiple parameter combination can lead us to a proper response to unpredictable loads. However, adaptive capacity is one of the fundamental appended values of a bio-inspired auto-scaling system. A matching among real scaling algorithms and bio-inspired, using real-world load series, allowed us to identify the suitability for specific workloads. In two cases Bio-inspired are better: in response to an not predictive and no temporal load or when the requests produce outliers (sudden load increases).

The model introduced exhibits an over-provisioned behavior, with a trend to exceed the requirements, a situation which usually implies a higher cost. There is no cost for creating/stopping actions for software containers in some container services such as Amazon Elastic Container Service (ECS). Thus, Bio model is not penalized in this area since only running resources involve a cost. Indeed, a higher number of nodes for an extended period of time is costly. Our proposal limits peak time and at the same time reduces application response time. More evidences are required to confirm a cost increase, if it is produced.

Finally, additional research lines arise as a result of this work. The first goal is to enhance NOX functions, introducing predictive analysis, pricing models or green computation options. The second line would be focused on understanding and design optimized ways for cells/containers communications and mutual discovery, upgrading the ICM model.

REFERENCES

[1] (2018). *Dockers*. Accessed: Jan. 11, 2020. [Online]. Available: http://www.docker.com

[2] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: Yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*. Berlin, Germany: Springer, 2017, pp. 195–216.

[3] G. LLC. (2018). *Kubernetes*. Accessed: Oct. 1, 2018. [Online]. Available: https://kubernetes.io/

[4] A. S. Fundation. (2018). *Apache Mesos*. Accessed: Jan. 11, 2020. [Online]. Available: http://mesos.apache.org/

[5] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Mar. 2015, pp. 171–172.

[6] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-scaling Web applications in clouds: A taxonomy and survey," 2016, *arXiv:1609.09224*. [Online]. Available: http://arxiv.org/abs/1609.09224

[7] P. Hoenisch, I. Weber, S. Schulte, L. Zhu, and A. Fekete, "Four-fold auto-scaling on a contemporary deployment platform using docker containers," in *Proc. Int. Conf. Service-Oriented Comput.* Berlin, Germany: Springer, 2015, pp. 316–323.

[8] S. Binitha and S. S. Sathya, "A survey of bio inspired optimization algorithms," *Int. J. Soft Comput. Eng.*, vol. 2, no. 2, pp. 137–151, May 2012.

[9] A. K. Kar, "Bio inspired computing—A review of algorithms and scope of applications," *Expert Syst. Appl.*, vol. 59, pp. 20–32, Oct. 2016.

[10] F. D. Muñoz-Escoí and J. M. Bernabéu-Aubán, "A survey on elasticity management in PaaS systems," *Computing*, vol. 99, no. 7, pp. 617–656, Jul. 2017.

[11] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *J. Grid Comput.*, vol. 12, no. 4, pp. 559–592, Dec. 2014.

[12] Amazon. (2019). *ECS Auto-Scaling*. Accessed: Jan. 11, 2020. [Online]. Available: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/service-auto-scaling.html

[13] Google. (2019). *Google Auto-Scaling*. Accessed: Jan. 11, 2020. [Online]. Available: https://cloud.google.com/compute/docs/autoscaler/

[14] Microsoft. (2019). *Azure Autoscale*. Accessed: Jan. 11, 2020. [Online]. Available: https://azure.microsoft.com/en-us/features/autoscale/

[15] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Elasticity in cloud computing: State of the art and research challenges," *IEEE Trans. Services Comput.*, vol. 11, no. 2, pp. 430–447, Mar. 2018.

[16] G. Moltó, M. Caballer, and C. de Alfonso, "Automatic memory-based vertical elasticity and oversubscription on cloud platforms," *Future Gener. Comput. Syst.*, vol. 56, pp. 1–10, Mar. 2016. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0167739X15003155, doi: 10.1016/j.future.2015.10.002.

[17] Jelastic. (2019). *Jelastic Web Page*. Accessed: Jan. 11, 2020. [Online]. Available: https://jelastic.com

[18] P. P. Kukade and G. Kale, "Auto-scaling of micro-services using containerization," *Int. J. Sci. Res.*, vol. 4, no. 9, pp. 1960–1963, 2015.

[19] C. Kan, "DoCloud: An elastic cloud platform for Web applications based on docker," in *Proc. 18th Int. Conf. Adv. Commun. Technol. (ICACT)*, Jan. 2016, pp. 478–483.

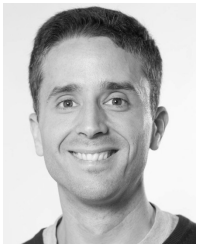
[20] N. M. Calcavecchia, B. A. Caprarescu, E. Di Nitto, D. J. Dubois, and D. Petcu, "DEPAS: A decentralized probabilistic algorithm for auto-scaling," *Computing*, vol. 94, nos. 8–10, pp. 701–730, Sep. 2012.

[21] A. Najjar, X. Serpaggi, C. Gravier, and O. Boissier, "Multi-agent negotiation for user-centric elasticity management in the cloud," in *Proc. IEEE/ACM 6th Int. Conf. Utility Cloud Comput.*, Dec. 2013, pp. 357–362.

[22] B. An, V. Lesser, D. Irwin, and M. Zink, "Automated negotiation with decommitment for dynamic resource allocation in cloud computing," in *Proc. 9th Int. Conf. Auto. Agents Multiagent Syst.*, vol. 1, 2010, pp. 981–988.

[23] T. C. Chieu and H. Chan, "Dynamic resource allocation via distributed decisions in cloud environment," in *Proc. IEEE 8th Int. Conf. e-Bus. Eng.*, Oct. 2011, pp. 125–130.
[24] S. Son and K. Mong Sim, "A price- and-time-slot-negotiation mechanism for cloud service reservations," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 42, no. 3, pp. 713–728, Jun. 2012.
[25] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Autonomic vertical elasticity of docker containers with ELASTICDOCKER," in *Proc. IEEE 10th Int. Conf. Cloud Comput. (CLOUD)*, Jun. 2017, pp. 472–479.
[26] P. D. Kaur and I. Chana, "A resource elasticity framework for QoS-aware execution of cloud applications," *Future Gener. Comput. Syst.*, vol. 37, pp. 14–25, Jul. 2014.
[27] D. Jiang, G. Pierre, and C.-H. Chi, "Autonomous resource provisioning for multi-service Web applications," in *Proc. 19th Int. Conf. World Wide Web (WWW)*, 2010, pp. 471–480.
[28] L. R. Moore, K. Bean, and T. Ellahi, "Transforming reactive auto-scaling into proactive auto-scaling," in *Proc. 3rd Int. Workshop Cloud Data Platforms (CloudDP)*, 2013, pp. 7–12.
[29] V. R. Messias, J. C. Estrella, R. Ehlers, M. J. Santana, R. C. Santana, and S. Reiff-Marganiec, "Combining time series prediction models using genetic algorithm to autoscaling Web applications hosted in the cloud infrastructure," *Neural Comput. Appl.*, vol. 27, no. 8, pp. 2383–2406, Nov. 2016.
[30] A. Hussain. (2014). *Performance of Docker vs VMs*. Accessed: Jan. 11, 2020. [Online]. Available: https://es.slideshare.net/Flux7Labs/performance-of-docker-vs-vms
[31] V. Gupta, K. Kaur, and S. Kaur, "Performance comparison between light weight virtualization using docker and heavy weight virtualization," *Int. J. Adv. Technol. Eng. Sci.*, vol. 1, no. 5, pp. 509–514, 2017.
[32] B. B. Rad, H. J. Bhatti, and M. Ahmadi, "An introduction to docker and analysis of its performance," *Int. J. Comput. Sci. Netw. Secur.*, vol. 17, no. 3, p. 228, 2017.
[33] J. Von Neumann and A. W. Burks, *Theory Self-Reproducing Automata*. Urbana, IL, USA: Univ. of Illinois Press, 1996.
[34] J. Byl, "Self-reproduction in small cellular automata," *Phys. D, Nonlinear Phenomena*, vol. 34, nos. 1–2, pp. 295–299, Jan. 1989.
[35] M. Gardner, "Mathematical games: The fantastic combinations of John Conway's new solitaire game 'life,'" *Sci. Amer.*, vol. 223, no. 4, pp. 120–123, 1970.
[36] A. Deshpande and M. Kumar, *Artificial Intelligence for Big Data: Complete Guide to Automating Big Data Solutions Using Artificial Intelligence Techniques*. Birmingham, U.K.: Packt, 2018.
[37] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, vol. 53. Berlin, Germany: Springer, 2003.
[38] S. Salzberg, D. Searls, and S. Kasif, "Computational methods in molecular biology," in *New Comprehensive Biochemistry*. Amsterdam, The Netherlands: Elsevier, 1999.
[39] R. Khurana, *Software Engineering (WBUT)*, 4th ed. Vikas Publishing House, 2016, ch. 8.
[40] J. Herrera. (2018). *Cobeats—Container Bio-Inspired Enhanced Autoscaling System*. [Online]. Available: https://github.com/grycap/cobeats
[41] W. Ai, K. Li, S. Lan, F. Zhang, J. Mei, K. Li, and R. Buyya, "On elasticity measurement in cloud computing," *Sci. Program.*, vol. 2016, 2016, Art. no. 7519507, doi: 10.1155/2016/7519507.
[42] A. Kochut and K. Beaty, "On strategies for dynamic resource management in virtualized server environments," in *Proc. 15th Int. Symp. Modeling, Anal., Simulation Comput. Telecommun. Syst.*, Oct. 2007, pp. 193–200.
[43] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-26, no. 1, pp. 43–49, Feb. 1978.
[44] Amazon. (2018). *Amazon Step Scaling*. Accessed: Jan. 11, 2020. [Online]. Available: https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scaling-simple-step.html
[45] M. McNaughton, "Predictive pod auto-scaling in the kubernetes container cluster manager," M.S. thesis, Williams College, Williamstown, MA, USA, 2016, p. 49. [Online]. Available: https://github.com/rootsongjc/cloud-native-slides-share/blob/master/kubernetes/Predictive-Pod-Auto-scaling-in-the-Kubernetes-Container-Cluster-Manager-by-Matt-McNaughton.pdf
[46] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: A view from the edge," in *Proc. 7th ACM SIGCOMM Conf. Internet Meas. (IMC)*, 2007, pp. 15–28.
[47] (1995). *NASA*. Accessed: Jan. 11, 2020. [Online]. Available: ftp://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html
[48] (1998). *World Cup Web Site Access Logs*. Accessed: Jan. 11, 2020. [Online]. Available: ftp://ita.ee.lbl.gov/html/contrib/WorldCup.html
[49] C. Henderson, *Building Scalable Web Sites*. Newton, MA, USA: O'Reilly Media, 2006.
[50] OpenShift. 2013. *Best Practices for Horizontal Application Scaling*. Accessed: Jan. 11, 2020. [Online]. Available: https://blog.openshift.com/best-practices-for-horizontal-application-scaling/
[51] Nomad. (2019). *The Million Container Challenge*. Accessed: Jan. 11, 2020. [Online]. Available: https://www.hashicorp.com/c1m
[52] D. Blog. (2015). *Scale Testing Docker Swarm to 30,000 Containers*. Accessed: Oct. 1, 2020. [Online]. Available: https://blog.docker.com/2015/11/scale-testing-docker-swarm-30000-containers/
[53] O. Docker. (2016). *Evaluating Container Platforms at Scale*. Accessed: Jan. 11, 2020. [Online]. Available: https://medium.com/on-docker/evaluating-container-platforms-at-scale-5e7b44d93f2c
[54] Kubernetes. (2015). *Kubernetes Performance Measurements and Roadmap*. Accessed: Jan. 11, 2020. [Online]. Available: https://kubernetes.io/blog/2015/09/kubernetes-performance-measurements-and

JOSÉ HERRERA received the master's degree in parallel and distributed computing from the Universitat Politècnica de València (UPV). He is currently pursuing the Ph.D. degree with the Grid and High Performance Computing Research Group (GRyCAP). He worked as a Predoctoral Researcher with the National Institute of Informatics (Tokyo), in 2016. He is also a Software Engineer with the Business Intelligence Team, Regional Government of Valencia (DGTIC-GVA), Spain. His research interests include big data infrastructures, scalability, business intelligence, and cloud computing.



GERMÁN MOLTÓ received the B.Sc. and Ph.D. degrees in computer science from the Universitat Politècnica de València (UPV), Spain, in 2002 and 2007, respectively. He has been a member of the Grid and High Performance Computing Research Group (GRyCAP), Institute for Molecular Imaging (I3M), since 2002. He is also an Associate Professor with the Department of Computer Systems and Computation (DSIC), UPV. He has participated in several H2020 European projects related to cloud computing (INDIGO-DataCloud, EOSC-HUB, DEEP Hybrid-DataCloud, and EOSC-SYNERGY) and led national research projects in the area of cloud computing. His broad research interests include cloud computing and scientific computing.