Value-Based Manufacturing Optimisation in Serverless Clouds for Industry 4.0

Piotr Dziurzanski
Department of Computer Science,
University of York
York, UK
piotr.dziurzanski@york.ac.uk

Jerry Swan
Department of Computer Science,
University of York
York, UK
jerry.swan@york.ac.uk

Leandro Soares Indrusiak
Department of Computer Science,
University of York
York, UK
leandro.indrusiak@york.ac.uk

ABSTRACT

There is increasing impetus towards 'Industry 4.0', a recently proposed roadmap for process automation across a broad spectrum of manufacturing industries. The proposed approach uses Evolutionary Computation to optimise real-world metrics. Features of the proposed approach are that it is generic (i.e. applicable across multiple problem domains) and decentralised, i.e. hosted remotely from the physical system upon which it operates. In particular, by virtue of being *serverless*, the project goal is that computation can be performed 'just in time' in a scalable fashion. We describe a case study for value-based optimisation, applicable to a wide range of manufacturing processes. In particular, value is expressed in terms of Overall Equipment Effectiveness (OEE), grounded in monetary units. We propose a novel online stopping condition that takes into account the predicted utility of further computational effort. We apply this method to scheduling problems in the $(max, +)$ algebra, and compare against a baseline stopping criterion with no prediction mechanism. Near optimal profit is obtained by the proposed approach, across multiple problem instances.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; • **Applied computing** → **Supply chain management**; • **Computing methodologies** → *Genetic algorithms*;

KEYWORDS

Plant optimisation, Genetic Algorithm, Value Curve, Stopping Condition, Function as a Service, FaaS, Serverless Clouds.

ACM Reference Format:
Piotr Dziurzanski, Jerry Swan, and Leandro Soares Indrusiak. 2018. Value-Based Manufacturing Optimisation in Serverless Clouds for Industry 4.0. In *GECCO '18: Genetic and Evolutionary Computation Conference, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.1145/3205455.3205501

GECCO '18, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5618-3/18/07.
https://doi.org/10.1145/3205455.3205501

1 INTRODUCTION

The 'Industry 4.0' concept[1] envisions increasingly automated manufacturing, characterized by the integration of Computational Intelligence methods into the production process. Properties typically associated with Industry 4.0 include a) interoperability of the cyber-physical components of the system and b) decentralized decision-making. Widespread adoption of Industry 4.0 will clearly not be possible if the Computational Intelligence methods require significant bespoke effort. The proposed methods must therefore exhibit both a high degree of cross-domain genericity and also require minimal end-user expertise to be applied to some variant application domain. Relative to other optimisation methods, these are both advantages enjoyed by Evolutionary Computation.

In this article, we present a case study of a distributed 'Function as a Servce' (FaaS) system that uses Evolutionary Computation to perform scalable optimisation. We address a real-world issue that is frequently neglected in many traditional benchmarks: the effect that time spent optimising has on overall manufacturing profit (OEE). We present a novel stopping condition which takes this into account. We define a model that combines manufacturing profit/loss with the predicted value of further computation. To obtain both genericity and real-world grounding, combined model values are expressed in terms of monetary units (sometimes termed '$EE' instead of OEE [17]).

The central notion of Industry 4.0 is that, by being rapidly responsive to the dynamic arrival of manufacturing orders, customers can then require only several units of a highly customised product [6]. They will have a set of highly configurable machines with automated material handling systems and a cloud-based management system [3]. Such a service-oriented manufacturing model will also aim to maximise the profit from the plant by sharing manufacturing resources across a number of manufacturing orders [15].

An ubiquitous optimisation problem in smart factories is the allocation of manufacturing resources over time, while satisfying constraints in terms of time and cost [15]. But even a single machine can be configured in multiple ways, depending on the required manufacturing schedule (priorities, delivery time, etc) and sustainability constraints (consumables and/or energy-saving conditions) [4]. Hence, optimisation is naturally interleaved with the manufacturing process in an online manner. This motivates the proposed approach of scalable optimisation with a grounded stopping criterion.

[1] http://www.plattform-i40.de/I40/Navigation/DE/Home/home.html

As all these parameters are chosen with the implicit aim of maximizing profit, value-based heuristics may be perceived as particularly suitable to solve this optimisation problem. Value-based heuristics have been shown to be beneficial in previous studies (e.g. in case of HPC system overload [28]). With such heuristics, a certain value is associated with each optimisation task. This value represents the importance level of the task and is usually proportional to the benefit accruing to the end-user from task completion. In the proposed system, we associate a so-called *value curve* with each manufacturing order, describing the temporal aspect of the yielded profit. The utility of an optimisation is therefore intrinsically time-dependent: completion of the optimisation process at a time point corresponding to a high value on the curve prevents the plant from becoming idle, and thus can increase the overall profit even if a better solution (viewed merely in terms of the optimisation task itself, rather than that overall profit) could *subsequently* be found.

A typical scheduling problem in smart factories requires substantial computing resources each time the factory needs reconfiguration, e.g. on arrival of a new manufacturing order. As these resources are needed on demand, cloud based optimisation using Genetic Algorithms (GA) is proposed in this paper. Evolutionary algorithms have been shown to be particularly effective in such applications [5] and performing the optimisation process in clouds can decrease the related costs [16]. To reduce the optimisation cost even further, the *Function as a Service (FaaS)* serverless cloud computing model has been chosen. Using this architecture, customers not only do not have to maintain the computing infrastructure, but are billed for the actual execution time of the computing resources, which are expected to become available within milliseconds of request. Thanks to these properties, the proposed optimisation scheme can be highly scalable depending on the size of the optimisation problem and timing constraints. Each of these possibilities is investigated in this paper.

Contribution: We propose a new modeling technique for manufacturing orders. This technique benefits from (i) the *a priori* knowledge of the dynamic value stemming from completion of the optimisation process at certain time, (ii) predicted utility of further optimisation, (iii) scalable computing resources available on demand in the serverless cloud computing architecture.

It is typical in optimisation research for the trade-off between solution quality and execution time to be implicit, with the latter often being expressed in terms of a coarse measure such as 'number of evaluations of the objective function'. This is of course good scientific practice, but is not sufficient to capture the needs of our real-world application. The second contribution therefore employs a value curve to make this trade-off completely explicit, grounding computational processing in terms of monetary profit, as described earlier.

2 RELATED WORK

In this Section, we discuss previous work related the three combined aspects of this paper: i) GA stopping criteria, ii) value-based heuristics, iii) evolutionary optimisation in the cloud:

Stopping Criteria

According to Michalewicz [19], the most common stopping criteria of a GA are either a) statically-determined upper limits on the number of generations or fitness function evaluations or b) dynamic prediction of further improvement based on genotypic and/or phenotypic convergence. Safe et al [21] argue that dynamic prediction is preferable. One of such alternatives has been proposed by Hernandez et al [12], with an adaptive stopping criterion which was experimentally shown to stop at optimal solutions with a high probability. In Hajji et al [10], a stopping criterion based on an approximation of the objective function has been compared with criteria based on both genotypic and phenotypic convergence. The genotypic convergence was evaluated by comparing the percentage convergence of each gene against a certain threshold. The phenotypic convergence was evaluated using two metrics: *online performance* converges to a stable value when the solutions converge and *offline performance* converges when the probability of improving the solution decreases. The approximation-based method gave the best results of the proposed approaches, but at the expense of greatest computation time. The online and offline performances were much faster and close to the approximation-based ones with regard to quality. The genotypic criteria were shown to be not efficient. Consequently, in our proposed approach only phenotypic convergence is considered.

In Yin et al [29], the Standard Deviation (SD) of fitness values is employed as a stopping criterion, terminating the optimisation process when SD is lower than a given threshold. In that paper, an SD-based stopping criterion has been shown to speed up the convergence and shorten the search time for scheduling independent tasks in a grid environment. This criterion is also used as a part of our proposed stopping criterion.

Perroni et al focus on an estimation of a beneficial stopping point for any swarm-based search algorithm [20]. In that paper, a sequence of auto-adapted exponential and log-like curves are proposed to model the algorithm convergence. In this paper, a similar approach based on the rational function extrapolation [25] is used to predict future fitness values and thus apply a value-based heuristic to trade between the potential value gain due to a better solution quality and the loss caused by the longer computation.

Value Based Heuristics

The main goal of value-based heuristics is to inform a decision process that maximises overall value to end-user [1]. Several value-based heuristics have been employed to allocate processing tasks. Theocharides et al [27] assumed task values to be fixed, whearas Burns et al [2] allow task values to change over time. In the latter case, the value can be described with a so-called *value curve*, a function whose domain represents the computation time (with origin at the release time of the process/container), whereas the codomain represents the values themselves [13]. Typically, a value curve is non-increasing and reaches a value of zero at a certain time point. After that point, there is no benefit from computing the task and it can be dropped to avoid consuming unnecessary processor resource. During each scheduling event, a task with the highest value at that moment can be selected, as discussed in Theocharides et al. The innate risk of such technique is to select a task with large

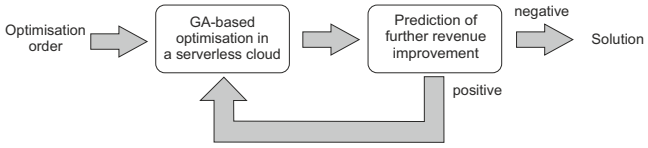

Figure 1: General scheme of the proposed approach

outlying resource requirements, where a set of less computationally expensive tasks may actually be preferable. Burkimsher et al [1] proposed to first allocate tasks with maximal remaining value, calculated as the area under the value curve from the current time to the zero point. A number of value-related heuristics have been compared by Singh et al [24], highlighting the benefits originating from the access to historic execution data. In this paper, a similar assumption is made: the estimated execution time of a GA invocation (termed a 'stage', hereafter), based on historically measured cases, is used to decide whether to terminate stage sequencing.

Value-based allocation of Docker containers have been proposed by Dziurzanski et al [8]. However, the authors of that paper focused on allocation of multiple independent containers to maximise the cumulative value, whereas the goal of this paper is to maximise the profit obtained from a single manufacturing order. Moreover, the related execution cost was not considered in that paper. Similarly, containers were executed in a local cluster, with customised scheduler installed on each node, whereas we instead focus on execution using serverless clouds.

A complementary notion to the value curve, entitled 'price curve', has been used in Henzinger et al [11] to present a varied cost of computing resources. In contrast, we assume constant cost for executing an optimisation engine in a cloud, as this is case for the major cloud vendors. We share with Henzinger the expression of value curves in monetary units.

Cloud-based Evolutionary Computation

A recent position paper [22] presents a conceptual workflow for the deployment and execution of distributed GAs. The software container technology (Docker) and a lightweight Linux distribution created to execute containers (CoreOS) has been used for large and scalable deployments on different infrastructure, focusing on security, consistency and reliability. This idea has been further extended in Salza et al [23], where evolutionary machine learning classifiers have been deployed to the cloud. In the proposed solution, a similar architecture is used for performing evolutionary optimisation. In particular, Docker containers are used to execute instances of jMetal [7], a popular Java framework offering a variety of algorithms for single- and multi-objective metaheuristic optimisation.

In Ma et al [16], a master-slave topology implementing a distributed evolution algorithm was employed. The master assigned the individuals from each generation to the slave nodes based on their load information and then collected the corresponding fitness values. The comparison with allocation of the same number of individuals to each node has been conducted for 32, 48 and 64 nodes. The obtained improvement of the computation time has ranged from 6% to 39% depending on the cluster size. While shortest time was achieved for the largest case, the strategy proposed in that paper has not considered heterogeneous architecture or various communication costs. The proposed solution also takes communication overhead into consideration. Since our approach is serverless, the number of nodes is decided dynamically.

Leclerc et al [14] propose a cloud-based framework facilitating large scale evolutionary experiments. Their framework provides a master-slave architecture, with nodes communicating via JSON over HTTP. The applied scheduling policy aims to uniformly spread the load across peers. The slave node with minimal load is chosen for each incoming fitness function evaluation task. There is no possibility of sharing processing units between tasks. Consequently, if there is no slave with an idle processing unit, the task is placed in a FIFO queue. To guarantee the appropriate amount of computing resources, the framework is intended to be executed on virtual machines (VMs) whose number is steered by the cloud provider, using facilities such as Amazon Auto Scaling Group. When the smoothed expected time to empty the queue is larger or smaller than certain thresholds, a VM is added or removed, respectively.

In contrast, in an approach advocated by a recent position paper [26], the framework presented in this paper executes a GA on several machines in accordance with the *serverless* computing paradigm. The motivation for the serverless approach is to provide both scalability and cost-effectiveness: payment is made only for actual computation performed.

From this literature survey, it follows that there is no prior work on stopping criteria for maximizing increase the overall benefits of an optimization process that is itself costly. This problem is investigated in this paper, the general scheme of which is illustrated in Fig. 1.

3 SYSTEM ARCHITECTURE AND PROBLEM DESCRIPTION

The class of optimisation problems analysed in this paper concern manufacturing plants. The value gained by an end-user from the optimisation depends on both solution quality and the time taken by the optimisation process itself. Since the optimisation process is performed by a serverless cloud, the system architecture covers the problem domain model and the cloud configuration.

In the following, we further describe the two components of Fig. 1. For the considered case study, the left-hand component corresponds to the optimisation of manufacturing plant that is specified via the $(max, +)$ algebra. In the right-hand component, the stopping criterion is applied to the iterated application of the optimisation process in order to maximise overall profit.

3.1 Plant Optimisation

The plant model used in this paper is based on *max-plus algebra*, a discrete algebraic system in which the *max* operation takes the role of addition ($\oplus$) and the traditional addition operator instead takes on the role of multiplication ($\otimes$). The max-plus algebra is convenient for modeling discrete event systems, since the basic operations of such systems, such as temporal transitions and synchronisation, can be described with a set of simple linear equations [9].

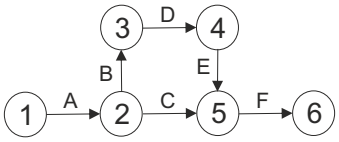

Figure 2: Activity on Arrow representation of a plant

A simple example of a plant is presented in Fig. 2 using the 'Activity On Arrow' (AOA) notation. In this notation, the states ($1, \ldots, 6$ in Fig. 2) represent synchronisation points whereas the actual manufacturing activities (a.k.a. processes) are performed on traversal between states via arrows ($A, \ldots, F$ in Fig. 2). In this example, a manufacturing process begins at time t_1 in state 1. The first manufacturing process is performed during transition A between states 1 and 2, which lasts for d_A time units. Thus, state 2 is visited at $t_2 = t_1 \otimes d_A = t_1 + d_A$. Similarly, $t_3 = t_2 \otimes d_B$ and $t_4 = t_3 \otimes d_D$. The production process represented by arrow F can start after both C and E are completed, so the max operator is applied, $t_5 = (t_2 \otimes d_C) \oplus (t_4 \otimes d_E)$. Finally, the last manufacturing process, represented by arrow F, is finished at $t_6 = t_5 \otimes d_F$.

In practice, the processing time in each manufacturing process is not constant, as machines can operate in various *modes* [4], for example *full performance* or *eco* modes. The optimisation process then includes not only the assignment of jobs to machines, but also the selection of the mode that minimises production cost, thereby finding a compromise between processing time and dissipated energy.

As discused above, in the considered optimisation problems, both solution quality and optimisation time are relevant to the end-user. The value stemming from the later is described by a value curve VC [2, 13]. This curve is expressed in a monetary unit (e.g. GBP). A value curve is usually a monotonically decreasing function. Its highest value equals to V_{max} from the manufacturing order arrival up to the deadline of the manufacturing order scheduling. Then it trends towards zero with the increasing completion time due to penalty, for example as shown in Fig. 3, where the value curve VC of manufacturing order O assumes its maximal value from the arrival time of O, AT, to the deadline of the optimisation of O, D. The optimisation time of manufacturing order O lasts from AT to ET. The total income of an end-user depends partially on the value of the value curve at ET.

The reduction in the manufacturing order value due to delay can be determined by observing the value of the value curve at the delayed completion time. A long optimisation time may result in zero value and thus the job becomes worthless to its end-user. Further, the cost of this optimisation can be considered as a loss. Therefore, the manufacturing order may be rejected if zero or a negative value is expected after completing it.

3.1.1 GA encoding, operators and fitness. The underlying optimisation problem considered in this case study is the configuration of a manufacturing process, specified by the *sequencing* of a fixed number of *machines*, each with a fixed number of *operating modes*. The corresponding geneome is therefore an integer-based representation, derived from the structure of the AoA network representing

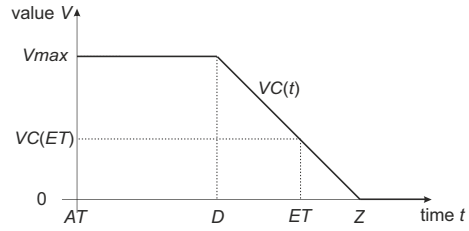

Figure 3: An example value curve of manufacturing order O

the plant, e.g. as shown in Fig. 2. The genome consists of a sequence of pairs (m, o) for machine m and operating mode o. There is one such pair for each arrow in the corresponding AoA representation.

The chosen GA operators are the familiar random mutation, one-point crossover and selection, with mutation probability 0.01 and crossover probability of 0.7, these values being obtained after a small amount of manual tuning. The population size has been fixed to 500 in all the experiments.

The fitness of a configuration is given as a weighted sum of the *makespan* and *energy* dissipated, each of which are a function of both machine placement in the AoA graph and the associated mode of each machine. It has the same monetary unit as the given value curve VC of the optimised problem.

3.2 Prediction of Revenue Improvement

The optimisation is performed during a number of stages. During the i-th stage $i \in \{1, \ldots n\}$, GA iterations are executed in parallel. Then the results are gathered and a stopping condition is checked, based on the prediction of the total value improvement in the subsequent stage.

We now describe the parameterisation of the case study considered in this paper.

3.2.1 Problem parameters. Symbols and abbreviations used are summarised in full in Table 1. Each problem instance is parameterised as follows:

- Input: manufacturing order O including: the plant given in the AoA form, its value curve VC and arrival time AT, a potentially unbounded number of slave processing nodes with (monetary) execution cost per time unit β and the number of individuals sent to each processing node.
- Objective: Maximise the profit obtained from the manufacturing order.

The profit from the manufacturing order depends on the following factors:

- the fitness value returned by the GA,
- total processing time allocated to the optimisers,
- cloud processing cost (per container invocation).

4 PROPOSED APPROACH

We now proceed to describe the proposed approach.

4.1 Value curve

The value curve models the value of a process to its end-user as a function of time, $VC(t)$. It may assume various shapes, as discussed

Table 1: Symbols and abbreviations used in the paper

Symbol	Description
O	manufacturing order
VC	value curve
AT	manufacturing order arrival time
D	manufacturing order optimisation deadline
Z	manufacturing order zero value time
V_{max}	maximal value of the manufacturing order value curve
f_i	minimal fitness function value after the i-th stage
$\hat{f}_i$	a predicted value of f_i
β	cost of a container execution per time unit (constant)
p_i	no. of containers executed during the i-th stage
t_i	time of computing of the i-th stage (measured)
c_i	cost of computing the i-th stage
T_i	cumulative time of computing the first i stages
C_i	cumulative cost of computing the first i stages
P_i	profit generated after computing the i-th stage
$\hat{P}_i$	prediction of P_i
$\hat{t}_i$	prediction of t_i
$\hat{c}_i$	prediction of c_i
I_i	income obtained after the i-th stage
sd_i	standard deviation of the population after stage i

in Burkimsher [1]. For the proposed approach, the shape shown in Fig. 3 has been chosen, which models generating the maximum value (e.g. as agreed in a contract) up to a certain deadline, after which a certain penalty is imposed every time unit. This shape can be intuitively explained as up to the deadline, the factory is occupied with other, previously configured manufacturing orders. So the deadline is the earliest time the factory can start manufacturing new products. Thus, there is no extra benefit in computing a new configuration well before the deadline, but after the deadline the factory becomes idle until a new configuration is found. As during this idle interval both relative overhead cost (ROC) and relative direct labor cost (RDLC) are incurred proportional to the idle time, the value of the solution decreases [17]. Without any further modification of the proposed approach, this shape can be exchanged with any other non-increasing function if a curve better describing a certain process is identified.

The chosen value curve assumes positive values starting from the time of the manufacturing order arrival, AT. As in this paper we consider only a single order scenario, without any loss of generality it may be assumed that $AT = 0$. The maximum value of $VC(t)$ is equal to V_{max} and is observed from AT to a certain deadline, $D > AT$. Finally, $VC(t)$ assumes zero value from zero value time, $Z > D$. This shape of the value curve can be described with the following equation

$$VC(t) = \begin{cases} V_{max} & \text{for } AT < t \leq D, \\ \frac{-V_{max}}{Z-D}(t-D) + V_{max} & \text{for } D < t \leq Z, \\ 0 & \text{for } t > Z. \end{cases} \quad (1)$$

4.2 Time and cost of stage execution

The optimisation is performed in stages until the applied stopping condition is satisfied. The stage index is denoted with i, $i \in \mathbb{N}$. During the i-stage, the optimisation is performed on p_i slave nodes. As these nodes are executed in the FaaS manner, the monetary cost of using them is given by value β per second for each instance (for example, in IBM Cloud it was \$0.000017 per second of execution, per GB of memory allocated on 21.01.2018). The maximal slave execution time in the i-th stage is equal to t_i. Thus the upper-bound on cost of the execution of this stage for container c_i is given by:

$$c_i = \beta \cdot t_i \cdot p_i. \quad (2)$$

The cumulative cost of computing the first i iterations, C_i, is equal to:

$$C_i = \sum_{j=1}^{i} c_j. \quad (3)$$

The predicted execution time of a stage, $\hat{t}_i$ is determined via the extrapolation mechanism described in Section 4.3. The manufacturing income yielded after the i-th iteration is a difference between the income given by value curve VT at the moment of completion the i-th stage and the manufacturing cost, described by fitness value f_i, i.e.

$$I_i = VC(T_i) - f_i. \quad (4)$$

The profit generated after execution of the i-th stage is expressed as a difference between the income and the cumulative cost of the optimisation:

$$P_i = I_i - C_i. \quad (5)$$

4.3 Value prediction

The values of t_i and f_i can be predicted via extrapolation. The extrapolation method used is the Bluirsch and Stoer algorithm [25], an extension of the well-known Neville interpolation/extrapolation algorithm to *diagonal* rational functions $p(x)/q(x)$ for polynomials p, q where p is of degree m (the length of the history vector from which to extrapolate) and the diagonal property requires that q is of degree m or $m + 1$, according as m is even. In many cases, this method can be analytically shown to provide superior accuracy to more traditional methods of polynomial extrapolation [25]. For history lengths of 3 or less, such extrapolation is either undefined or else the result was empirically determined to be inaccurate: the predicted value of f_i is then given by the best fitness found so far and that of t_i by the last (actual) processing time. After predicting the values $\hat{f}_i, \hat{t}_i$, they are used to predict the profit generated after the subsequent, $(i + 1)$-th stage as follows:

$$\hat{P}_{i+1} = VC(T_i + \hat{t}_{i+1}) - \hat{f}_{i+1} - C_n - \hat{c}_{i+1}. \quad (6)$$

This value can be used in a value-based stopping criterion, as described in the subsection below.

4.4 Stopping criteria

The stopping criteria are evaluated for a container at each stage i. We first apply an *absolute* criterion (ensuring that the process will eventually terminate) by comparing the i to a fixed upper bound on the number of stages (here, a value of 100 was empirically chosen). The *phenotypic convergence* criterion compares the Standard Deviation sd_i of the GA population against a threshold value (here, 0.02), similarly to e.g. Yin et al [29]. The *predicted profit* criterion uses the method of diagonal rational extrapolation described above to predict whether the execution of the subsequent stage will not decrease the profit generated by the optimised process or not:

$$P_n > \hat{P}_{n+1}. \quad (7)$$

The benefits of these stopping criteria are evaluated in Section 6.

5 IMPLEMENTATION ISSUES

Similarly to Leclerc et al [14], the proposed optimisation process is implemented using a master-slave paradigm: the master is executed locally and awaits manufacturing orders. Upon arrival of a manufacturing order, its role is to prepare an appropriate plant configuration scheme and generate a set of individuals for the GA-based optimisation. This data is sent to a certain number of slave nodes, where, at each stage, the actual GA-based optimisation algorithm is executed for a certain number of iterations. Finally, the results are returned to the master node which evaluates the stopping condition as described earlier in this paper.

The GA-based optimiser, executed remotely by the slave nodes, has been implemented within the jMetal framework and placed inside a Docker container [18]. This container acts as a REST-compliant Web service, awaiting input in the form of a population of proposed plant configurations (i.e. manufacturing workflows) to be optimised. After performing the stipulated number of GA iterations (see Section 6), the container returns a new generation of proposed plant configurations. Communication between the master and slave nodes is performed via JSON over HTTP.

As previously mentioned, since the slave nodes are stateless, they are not bound to a particular optimisation process and thus can be executed in accordance with the serverless computing paradigm. This means that the slave nodes are executed on demand without provisioning virtual machines. Slave nodes do not have to be active between consecutive invocations and thus the company is billed only for the real computation time of the slaves. One of the public vendors that offer 'on demand' execution of a Docker container is IBM OpenWhisk[2]. When an OpenWhisk Docker action is invoked by the master via a REST API call, OpenWhisk pulls the Docker image for the slave node from Docker Hub and then forwards the input HTTP POST request with the configuration and individuals. After finishing the computation, the request responds with the resulting population of configurations and the slave node is killed. Alternatively, Apache OpenWhisk[3] can be also used in a private cloud or a public cloud provided by other vendors.

6 EXPERIMENTAL RESULTS

To evaluate the proposed optimisation approach, we first describe the application to a single large manufacturing plant. We then consider a larger number of problem instances.

6.1 A larger problem instance in-depth

The selected plant is representative of the larger instance sizes encountered when the system is coupled to the real-world equipment of the project's industrial partners. It is described by an AoA instance with 22 nodes, 6 levels and 43 arrows. Each manufacturing process can be executed using one of 8 machine types, each having from 1 to 9 operating modes with different performance and energy dissipation. The manufacturing cost depends on the selected machines and their modes, as described earlier. The search space for such an instance is too large to be realistically solvable by exhaustive methods without incurring overall monetary loss due to increased optimisation time.

[2]https://console.bluemix.net/openwhisk/
[3]https://openwhisk.apache.org/

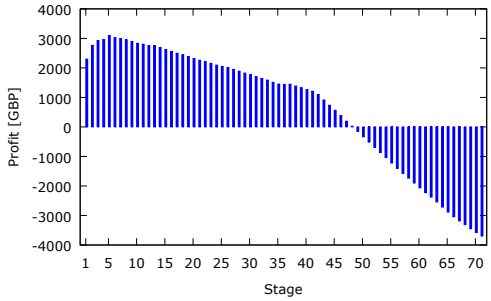

Figure 4: Profit yielded after each stage of the example plant with computation cost β = 0.5GBP

The value curve given for this particular instance is consistent with equation (1), with assumed parameters $AT = 0$, $D = 500$s, $Z = 1000$s, $V_{max} = 5000$GBP. One second of computations is assumed to cost $\beta = 0.5$GBP (lower values of this parameter are applied later in this section) and the initial number of containers run in parallel is set to $p_1 = 10$. In each stage, a fixed number of generations of a GA is executed.

As described above, the following **baseline** dynamic stopping criteria are applied to each container: (i) the Standard Deviation of population fitness after the ith stage, $sd_i \leq 10^{-6}$ *or* (ii) minimal fitness function value f_i was not improved during the previous 20 stages *or* (iii) the number of stages $i = 100$, used as a guarantee for the computation ending. The proposed criteria differs from this baseline by the additional inclusion of profit prediction.

During execution, criterion (i) stopped the continuation in a certain container after the 34th stage for the first time. So, after this stage, 9 parallel containers continued the execution (i.e. $p_{35} = 9$) up to stage 46th, where another container stopped computation and so on. Finally, as many as 71 stages have been computed and during the last stage only two containers continue the optimisation process ($p_{71} = 2$).

After each ith stage, the fitness function value to be computed in the next (i + 1-st) stage is predicted as described in subsection 4.3. The average prediction error was circa 2%. This accurate prediction can be well exploited by the proposed value-based stopping criteria, as discussed later.

The profit P_i obtained after ending computation at each ith stage is presented in Fig. 4. The highest profit is obtained after relatively early $i = 5$th stage. After this point, due to increasing computation cost and the decreasing slope of the value curve after the 40th stage, the yielded profit is significantly lower and beyond the 49th it becomes negative. Clearly, the baseline stopping criterion triggers too late. This is in contrast to the criterion proposed in equation (7). After applying this criterion, the profit is predicted to decrease after the 6th stage, which is the second best during the whole analysed range and only 2% worse than the highest possible profit.

In the previous example, a rather high cost of performing computation has been assumed. Let us compare these results with the second extreme case presented in Fig. 5, when the computation is performed for free, i.e. $\beta = 0$. In this case, the proposed stopping criterion from equation (7) terminates the execution after the 41st stage, which yields the highest possible profit. After this stage, the

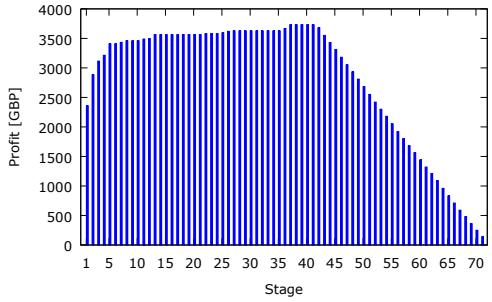

Figure 5: Profit yielded after each stage of the example plant with no computation costs

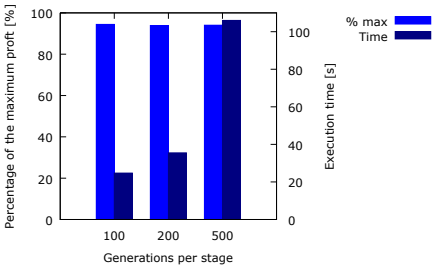

Figure 6: Profit and execution time obtained for containers executing assorted iteration numbers

profit drops due to the decreasing slope of the associated value curve.

6.2 Granularity

In this experiment, different numbers of GA generations, namely 100, 200 and 500, have been executed at each stage, during each container invocation. The experiment has been conducted for 10 different manufacturing orders, with representative characteristics, with the number of required manufacturing processes ranging from 18 to 59. The averaged results are presented in Fig. 6. As it is visible in the graph, the analysed granularity levels have no influence on the profit yielded by algorithm, as in all three cases it is almost equal to the maximal achievable profit from the given plants. However, the execution times of the considered optimisation processes differ significantly as it is more than 4 times longer for 500 generations per stage than 100 generations per stage.

6.3 Scalability

The proposed approach benefits from a serverless cloud execution, so that the number of containers executed in parallel can be easily scaled. The number of containers executed in parallel does not influence the total computation time, but it increases the total computation cost, as each second of container computation costs β.

Fig. 7 visualises the profit obtained from the optimisation of the plant described in subsection 6.1. In this experiment, various numbers of containers computing in parallel, from 1 to 10, were tested, with computation costs ranging from $\beta = 0$ to $\beta = 0.5$GBP. Despite the fact that 'number of containers' is discrete, a 3D surface

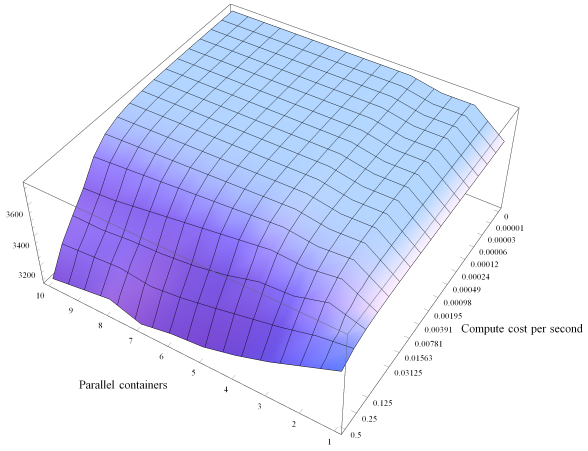

Figure 7: Computation cost per second by parallelisation

plot has been used to facilitate observation. It is worth noting that the β axis is expressed in a logarithmic scale (excepting the boundary case $\beta = 0$), as typical serverless computation cost is expected to be close to 0.00005GBP, but other orders of magnitude are added to cover a wider range of cloud architectures.

From this figure, it follows that the highest profits are yielded in the middle of the analysed range, for $p_i = 4$. This value can be then treated as a trade-off between the benefits of parallel execution, i.e. evolving the best results independently by a few optimisation processes and the increased monetary cost by executing a higher number of containers in parallel. But even in case of lower execution costs (including the extreme case $\beta = 0$), no additional profit is yielded by scaling p_i beyond 4.

Since using the larger number of containers increases costs, the higher standard deviation of the yielded profit considering various β has been observed for the largest number of the containers run in parallel $p_i = 10$. This value decreases almost linearly up to $p_i = 4$, for which standard deviation of the yielded profit is almost 10 times lower than for $p_i = 10$. So, if no parallelism is applied, the computational cost β is less important.

6.4 Stopping criteria

In order to compare the proposed approach with the baseline stopping criterion, 30 manufacturing orders whose number of manufacturing process steps ranged from 18 to 59 have been optimised using both the approaches. The maximal possible income value from each of the manufacturing order is equal to 5000GBP.

The stopping criterion used for baseline comparison does not consider profits: it is triggered when the fitness function value has not been improved for a certain number of generations. This is in contrast with the proposed stopping criterion, which aims to maximise profit by stopping the optimisation when no further profit gain is predicted. Consequently, the optimisation of a manufacturing order is stopped much earlier. For the considered set of manufacturing orders, the optimisation process has been completed 18.5 times faster. Hence, during such significantly shorter time, the obtained fitness function values are, on average, 34% worse, as shown in the box plot in Fig. 8 left (lower is better). However, the

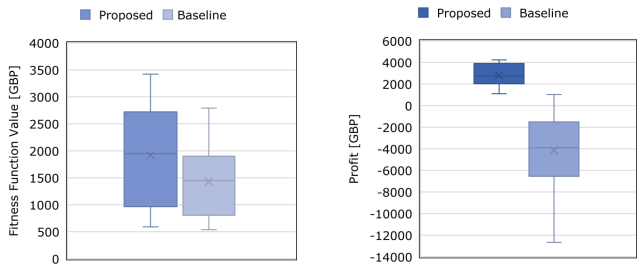

Figure 8: Comparison of the fitness function value (left) and profit (right) obtained with the proposed and the baseline stopping criteria

goal of the proposed method is to maximise the profit, which depends both on the fitness function value and the computation time. The impact of the latter results in the fact that using the baseline stopping criteria, 86% of the considered manufacturing orders lead to financial loss, whereas all of them are profitable when the proposed stopping criterion is applied. These profits are shown on the right of Fig. 8 (higher is better). Applying the proposed criterion leads to a cumulative profit of 83877GBP, whereas the baseline criterion lead to the negative profit of -124497GBP. Formal statistical comparison of the proposed and baseline profits for each problem instance (pairwise, via the Wilcoxon Signed Rank test) confirms significance (with p-value $1.9 * 10^{-9}$).

7 CONCLUSION

This article describes a serverless, cloud-based architecture that provides general and scalable support for the 'Just in Time' manufacturing process envisioned for 'Industry 4.0'. The architecture is equipped with a novel adaptive stopping criterion for optimising Overall Equipment Effectiveness (OEE), in which the predicted cost/benefit ratio of performing further optimisation is grounded in monetary units. The method was applied to a collection of representative case studies for optimal configuration of manufacturing plants, as specified via the $(max, +)$ algebra. We determined the most effective parallelisation strategy (implemented via stateless, Dockerised containers) and obtained near maximum profit from the resulting optimisation.

ACKNOWLEDGMENTS

The authors acknowledge the support of the EU H2020 SAFIRE project (Ref. 723634).

REFERENCES

[1] Andrew Burkimsher and Leandro Soares Indrusiak. 2016. Bidding policies for market-based HPC workflow scheduling. *CoRR* abs/1601.07047 (2016).
[2] Alan Burns, Divya Prasad, Andrea Bondavalli, Felicita Di Giandomenico, Krithi Ramamritham, John Stankovic, and Lorenzo Strigini. 2000. The meaning and role of value in scheduling flexible real-time systems. *Journal of systems architecture* 46, 4 (2000), 305–325.
[3] Baotong Chen, Jiafu Wan, Lei Shu, Peng Li, Mithun Mukherjee, and Boxing Yin. 2017. Smart Factory of Industry 4.0: Key Technologies, Application Case, and Challenges. *IEEE Access* (2017).
[4] Nancy Diaz, Elena Redelsheimer, and David Dornfeld. 2011. Energy consumption characterization and reduction strategies for milling machine tool use. *Glocalized solutions for sustainability in manufacturing* (2011), 263–267.
[5] Ulrich Dorndorf and Erwin Pesch. 1995. Evolution based learning in a job shop scheduling environment. *Computers & Operations Research* 22, 1 (1995), 25–40.
[6] Jocelyn Drolet, Colin L Moodie, and Benoit Montreuil. 1989. Scheduling factories of the future. *Journal of Mechanical Working Technology* 20 (1989), 183–194.
[7] Juan J Durillo and Antonio J Nebro. 2011. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software* 42, 10 (2011), 760–771.
[8] Piotr Dziurzanski and Leandro Soares Indrusiak. 2018. Value-Based Allocation of Docker Containers. In *The 26th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*.
[9] Hiroyuki Goto. 2014. Introduction to max-plus algebra. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. ACM, 21–22.
[10] Omessad Hajji, Stéphane Brisset, and Pascal Brochet. 2003. A stop criterion to accelerate magnetic optimization process using genetic algorithms and finite element analysis. *IEEE transactions on magnetics* 39, 3 (2003), 1297–1300.
[11] Thomas A Henzinger, Anmol V Singh, et al. 2010. A marketplace for cloud resources. In *Proceedings of the tenth ACM international conference on Embedded software*. ACM, 1–8.
[12] German Hernandez, Kenneth Wilder, Fernando Nino, and Julian Garcia. 2005. Towards a self-stopping evolutionary algorithm using coupling from the past. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. ACM, 615–620.
[13] Bhavesh Khemka, Ryan Friese, et al. 2015. Utility functions and resource management in an oversubscribed heterogeneous computing environment. *IEEE Trans. Comput.* 64, 8 (2015), 2394–2407.
[14] Guillaume Leclerc, Joshua E Auerbach, Giovanni Iacca, and Dario Floreano. 2016. The seamless peer and cloud evolution framework. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. ACM, 821–828.
[15] Yongkui Liu, Xun Xu, Lin Zhang, Long Wang, and Ray Y Zhong. 2017. Workload-based multi-task scheduling in cloud manufacturing. *Robotics and Computer-Integrated Manufacturing* 45 (2017), 3–20.
[16] Ning Ma, Xiao-Fang Liu, Zhi-Hui Zhan, Jing-Hui Zhong, and Jun Zhang. 2017. Load balance aware distributed differential evolution for computationally expensive optimization problems. In *GECCO Proceedings Companion, 2017*. ACM, 209–210.
[17] Darin Marcus and Laron Colbert. 2015. $EE, the Financial Aspect of OEE. (2015). https://www.qualitydigest.com/inside/operations-article/091715-ee-financial-aspect-oee.html
[18] Dirk Merkel. 2014. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.* 2014, 239, Article 2 (March 2014). http://dl.acm.org/citation.cfm?id=2600239.2600241
[19] Zbigniew Michalewicz. 2013. *Genetic algorithms + data structures = evolution programs*. Springer Science & Business Media.
[20] Peter Frank Perroni, Daniel Weingaertner, and Myriam Regattieri Delgado. 2017. Estimating stop conditions of swarm based stochastic metaheuristic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 43–50.
[21] Martín Safe, Jessica Carballido, Ignacio Ponzoni, and Nélida Brignole. 2004. On stopping criteria for genetic algorithms. *Advances in Artificial Intelligence–SBIA 2004* (2004), 405–413.
[22] Pasquale Salza, Filomena Ferrucci, and Federica Sarro. 2016. Develop, Deploy and Execute Parallel Genetic Algorithms in the Cloud. In *GECCO Proceedings Companion, 2016*. ACM, 121–122.
[23] Pasquale Salza, Erik Hemberg, Filomena Ferrucci, and Una-May O'Reilly. 2017. Towards evolutionary machine learning comparison, competition, and collaboration with a multi-cloud platform. In *GECCO Proceedings Companion, 2017*. ACM, 1263–1270.
[24] Amit Kumar Singh, Piotr Dziurzanski, and Leandro Soares Indrusiak. 2015. Value and energy optimizing dynamic resource allocation in many-core HPC systems. In *Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on*. IEEE, 180–185.
[25] J. Stoer, R. Bartels, W. Gautschi, R. Bulirsch, and C. Witzgall. 2002. *Introduction to Numerical Analysis*. Springer New York.
[26] Jerry Swan, Steven Adriaensen, et al. 2015. A Research Agenda for Metaheuristic Standardization. In *Proceedings of the Eleventh Metaheuristics International Conference (MIC), Agadir, Morocco*. https://goo.gl/kC06p5
[27] Theocharis Theocharides, Maria K Michael, Marios Polycarpou, and Ajit Dingankar. 2010. Hardware-enabled dynamic resource allocation for manycore systems using bidding-based system feedback. *EURASIP Journal on Embedded Systems* 2010 (2010), 3.
[28] Chee Shin Yeo and Rajkumar Buyya. 2006. A taxonomy of market-based resource management systems for utility-driven cluster computing. *Software: Practice and Experience* 36, 13 (2006), 1381–1419.
[29] Hao Yin, Huilin Wu, and Jiliu Zhou. 2007. An improved genetic algorithm with limited iteration for grid scheduling. In *Grid and Cooperative Computing, 2007. GCC 2007. Sixth International Conference on*. IEEE, 221–227.