Multi-Objective Optimization for Virtual Machine Allocation and Replica Placement in Virtualized Hadoop

Carlos Guerrero, Isaac Lera, Belen Bermejo, and Carlos Juiz, *Senior Member, IEEE*

Abstract—Resource management is a key factor in the performance and efficient utilization of cloud systems, and many research works have proposed efficient policies to optimize such systems. However, these policies have traditionally managed the resources individually, neglecting the complexity of cloud systems and the interrelation between their elements. To illustrate this situation, we present an approach focused on virtualized Hadoop for a simultaneous and coordinated management of virtual machines and file replicas. Specifically, we propose determining the virtual machine allocation, virtual machine template selection, and file replica placement with the objective of minimizing the power consumption, physical resource waste, and file unavailability. We implemented our solution using the non-dominated sorting genetic algorithm-II, which is a multi-objective optimization algorithm. Our approach obtained important benefits in terms of file unavailability and resource waste, with overall improvements of approximately 400 and 170 percent compared to three other optimization strategies. The benefits for the power consumption were smaller, with an improvement of approximately 1.9 percent.

Index Terms—Virtual machine allocation, file replica placement, hadoop, evolutionary computing and genetic algorithms

1 Introduction

Apache Hadoop is a common solution for implementing MapReduce for solving big data problems. Its data organization and distribution rely on HDFS, the Hadoop distributed file system [1]. HDFS is a distributed and scalable file system in which files are split into blocks (chunks) that are stored across the nodes of a cluster (DataNodes). Chunks are replicated in different DataNodes to guarantee data availability. When a MapReduce job is scheduled, Hadoop distributes the jobs among the DataNodes that place the chunks to be processed.

Hadoop can be deployed both in bare metal and virtualized datacenters. In the case of virtualized Hadoop, it deploys DataNodes in virtual machines (VMs), and the VM manager allocates these VMs in physical machines (PMs) [2], [3]. Virtualized Hadoop offers several benefits, such as easy cloning of images with lower operational costs, setting DataNodes on demand, reusing and sharing the physical infrastructure, and increasing resource utilization by consolidating multiple DataNodes on the same PM [4].

The policies for selecting the features of the VMs and distributing them in PMs and the chunk replicas in DataNodes are commonly known as VM template selection, VM allocation, and replica placement. Important questions arise in this distribution process: How many VMs are necessary to deploy the HDFS system? Which are the best features for the VMs? Which PMs should allocate the VMs? Where should the chunk replicas be stored? The management policies answer all these questions. An efficient implementation of these policies has a direct impact on the system performance and on the resource usages [5], [6]. However, the optimal solution cannot be directly calculated because it is an NP-hard problem and all the possible placement combinations should be measured [7].

Although VM allocation, VM template selection, and replica placement have been widely studied [5], [8], [9], [10], [11], [12], to the best of our knowledge, a simultaneous and coordinated solution for the three problems has not previously been proposed. Previous studies have focused on solving either the mapping of replicas in DataNodes [13], [14] or mapping the VM instances with their VM types and their allocations in PMs [15]. Our approach is a multi-objective optimization for power consumption, resource waste and file availability using a threefold management of VM allocation, VM type selection, and replica placement in virtualized Hadoop. We have adopted the non-dominated sorting genetic algorithm-II (NSGA-II), which is a genetic algorithm (GA) for multi-objective problems. Evolutionary approaches are common solutions for resource management in cloud environments [16]. We have considered several sizes for the experiments, and the results of our approach have been compared with three other scenarios based on the works of Long et al. [17] and Adamuthe et al. [18]. The main contributions of this article can be summarized as follows:

• *The authors are with the Computer Science Department, Balearic Islands University, Palma E07122, Spain.*
E-mail: {carlos.guerrero, isaac.lera, belen.bermejo, cjuiz}@uib.es.

Manuscript received 25 July 2017; revised 7 May 2018; accepted 14 May 2018. Date of publication 17 May 2018; date of current version 10 Oct. 2018. (Corresponding author: Carlos Guerrero.)
Recommended for acceptance by R. Prodan.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2018.2837743

(i) An approach for simultaneously managing VM allocation, VM template selection, and replica placement in virtualized Hadoop with the objective of minimizing the power consumption, physical resource waste, and file unavailability;
(ii) A formal definition for virtualized Hadoop;
(iii) An experimental validation of the optimization problem implemented with NSGA-II.

2 Related Work

The related work is organized in two parts: the first is evolutionary approaches for VM management, and the second is replica placement works in HDFS

Zhu et al. studied the optimization of VM scheduling in Amazon EC2 using a multi-objective evolutionary algorithm [19]. Gao et al. proposed an ant colony algorithm to optimize VM placement considering a dual objective: minimize the power consumption and physical resource waste. They demonstrated the competitiveness of their solution against multi-objective grouping genetic algorithms and single-objective approaches [20]. The same optimization objectives were taken into account in the work of Su et al. based on a firefly algorithm [21]. Kessaci et al. presented a Pareto multi-objective version of the energy-aware multi-start local search algorithm dealing with energy consumption and SLAs. The algorithm allocates the VMs by reducing the response time of the jobs inside the machines and the energy consumption in the physical level [22]. Lopez et al. [23] formulated a memetic algorithm for a many-objective VM allocation optimization of power consumption, network traffic, economical revenue, quality of service, and network load balancing. Xu et al. used fuzzy logic and grouping genetic algorithms to optimize the power consumption and thermal and resource efficiencies in VM placement [24]. Mi et al. proposed a genetic algorithm to self-reconfigure the allocation of VMs according to time-varying requirements and dynamic conditions [25].

Adamuthe et al. compared genetic algorithms with non-dominated sorting genetic algorithms (NSGA) to maximize physical resource usages, the balanced distribution of VMs among physical machines and the wasted resources [18]. Their work is probably the most similar work to our approach in terms of VM allocation.

There are several efforts addressing HDFS placement strategies. The most similar work to our approach in terms of replica placement is probably the work of Long et al. [17]. They proposed optimizing file unavailability, service time, load variance, energy consumption and access latency by managing the replication factor and replica placement in VMs. They used an improved artificial immune algorithm that they called multi-objective optimized replication management (MORM).

Basanta-Val et al. studied the requirements of time-critical applications for big data systems [26], and they proposed several patterns for real-time stream processing in big data to improve application performance [27].

Song et al. proposed a placement algorithm that optimizes the energy consumption [28]. Dai et al. proposed a policy that evenly distributes the replicas across the nodes in the data center to achieve a load-balanced configuration [29]. Eltabakh et al. [30] extended HDFS to allow applications to define and exploit customized placement strategies to improve the performance of the system. Maheshwai et al. proposed a reconfiguration of the block placements to turn cluster nodes on or off in terms of the average cluster utilization [31]. They obtained important benefits in terms of energy consumption. Cheng et al. [32] proposed a real-time event processing engine to detect the files with higher popularity and consequently to increase the replication factor. Their experiments showed improvements in reliability and performance. Wei et al. proposed a cost-effective replication manager for HDFS based on the use of B+ trees to improve cost and load balancing [33]. Wang et al. [34] proposed a placement and scheduling manager for Hadoop based on an evolutionary approach, the MOEA/D algorithm, to improve CPU and disk utilization. The benefits of interrelating elements from the VM allocation and file placement were studied in Lu et al. [35]. They proposed a decoupled architecture for Hadoop in which the storage of the files (HDFS) is performed in the PMs and the computation (MapReduce jobs) is conducted in the VMs. They defined an orchestrator to distribute the MapReduce jobs to the VMs allocated in the PMs that allocated the data to process and consequently reduce the data transfers.

3 Problem Statement and Formulation

3.1 System Modeling

The system is defined by (i) the characteristics of the physical machines, the virtual machines, and the HDFS file system; (ii) the allocation relationships between VMs and PMs, and the placement relationship between the replicas and VMs. Table 1 summarizes the model parameters that are explained in this section.

The system is modeled as a datacenter where each PM $pm_i \in PM$ is characterized by the capacity of its resources, $pmResCap_{pm_i}$; the power features, $pmPowFeat_{pm_i}$; and the failure metrics, $pmFail_{pm_i}$. The resource capacity is a vector that contains the capacities of each physical component. Our resource model is limited to three components—CPU, disk bandwidth and network bandwidth—but it could easily be extended by including as many elements as necessary. Therefore, $pmResCap_{pm_i} = \langle pmResCap^{cpu}_{pm_i}, pmResCap^{diskbw}_{pm_i}, pmResCap^{netbw}_{pm_i}\rangle$. The power features are represented by a vector that includes all the parameters of the power consumption model that we adopted [36] and whose details are explained in Section 3.3: $pmPowFeat_{pm_i} = \langle pmPowMin_{pm_i}, pmPowMax_{pm_i}, \alpha_{pm_i}, \beta_{pm_i}, \delta_{pm_i}, \gamma_{pm_i}\rangle$, where the first two are the minimum and maximum power consumptions of the PM, respectively, and the others are model coefficients. Finally, failures are modeled as a bathtub curve with respect to the CPU usage of the machine [37], $pmFail_{pm_i} = \langle pmFail^{min}_{pm_i}, pmFail^{max}_{pm_i}\rangle$.

Our scenario defines DataNodes as VM instances, and both concepts are used indistinctly throughout this article. DataNodes, or VM instances, $vm_n \in VM$, are characterized by their VM instance type, $vmt_t \in VMType$, defined through a non-injective and non-surjective function $vmtype : VM \rightarrow VMType$. The VM instance type defines the characteristics of the VM. In particular, it defines the VM instance's resource capacities, $vmResCap_{vm_n}$, and the VM instance's failure metrics, $vmFail_{vm_n}$. By considering the resource and failure

TABLE 1
Summary of the System Model Parameters

Parameter	Description
pm_i	Physical machine with id i
$pmResCap_{pm_i}$	Total capacity of the resource elements of the ith PM
$pmPowFeat_{pm_i}$	Power consumption model of the ith PM
$pmFail_{pm_i}$	Failure model of the ith PM
$pmResCon_{pm_i}$	Consumption of the physical resources of the ith PM
$hypResCon_{pm_i}$	Consumption of the physical resources consumed by the VM hypervisor
pmU_{pm_i}	Normalized resource utilization of the ith PM
vm_n	VM instance with id n
vmt_t	VM instance type with id t
$vmtype()$:	Relationship that determines the type of a VM instance
$vmResCap_{vm_n}$	Total provisioned capacity of system resources for the nth VM
$vmResCon_{vm_n}$	Consumption of the provisioned resources for the nth VM
$vmFail_{vm_n}$	Failure rate of the nth VM
$allocation()$:	Relationship for the allocation of VMs in physical machines
f_u	File with id u
fb_x^u	The xth chunk of the uth file
$fb_x^u[r]$	The rth replica of the xth block/chunk of the uth file
$fSize_{f_u}$	The size of the uth file
$fbSize_{f_u}$	Block/chunk size for the uth file
fC_{f_u}	Number of chunks in the uth file
fR_{f_u}	Replication factor for the uth file
$repResCon_{fb_x^u}$	Resource consumption generated in a DataNode each time a replica is accessed by an MR job
$repAccRate_{fb_x^u[r]}$	MR jobs' access rate for a replica
$placement()$:	Relationship for the storage of file chunks in VMs

models explained previously for the PMs, the VM capacity is defined as $vmResCap_{vm_n} = \langle vmResCap^{cpu}_{vm_n}, vmResCap^{diskbw}_{vm_n}, vmResCap^{netbw}_{vm_n}\rangle$, and equally for the failures, $vmFail_{vm_n} = \langle vmFail^{min}_{vm_n}, vmFail^{max}_{vm_n}\rangle$. For simplicity, throughout this manuscript, VM refers to VM instance, and template refers to VM instance type.

HDFS files are split into ordered pieces called file blocks or chunks. Thus, a file, $f_u \in HDFS$, is defined as the concatenation of its chunks, $f_u = fb_0^u \parallel fb_1^u \parallel fb_2^u \parallel .. \parallel fb^u_{fC_{f_u}-1}$, where fC_{f_u}, the number of chunks, is determined by the file size and the chunk size as $fC_{f_u} = \lceil \frac{fSize_{f_u}}{fbSize_{f_u}} \rceil$. HDFS allows setting the chunk size individually for each file, but it generally has the same value for all the files. To guarantee the availability of the files, HDFS replicates each chunk across several DataNodes. We define a file chunk as a set of replicas, $fb_x^u = \{fb_x^u(0), fb_x^u(1), \ldots, fb_x^u(fR_{f_u} - 1)\}$, where fR_{f_u}, the replication factor, is also set individually for each file.

The replicas are characterized by the resource consumption that the execution of MapReduce jobs (MR) generates, $repResCon_{fb_x^u[r]}$. Apache Hadoop distributes the jobs across the DataNodes to avoid moving the data [38]. Considering the scope of our resource model, $repResCon_{fb_x^u[r]} = \langle repResCon^{cpu}_{fb_x^u[r]}, repResCon^{diskbw}_{fb_x^u[r]}, repResCon^{netbw}_{fb_x^u[r]}\rangle$. The total workload in a DataNode depends on the access rate of each replica, $repAccRate_{fb_x^u[r]}$.

The storage of the replicas in the DataNodes is a many-to-one relationship modeled with a non-injective and non-surjective function, $placement : \{fb_x^u[r]\} \rightarrow VM$. Additionally, the VMs are deployed in PMs. This is also a many-to-one relationship modeled as a non-injective and non-surjective function $allocation : VM \rightarrow PM$.

The total resource consumption of a VM depends on the access rates of the replicas it places and on the consumption generated in those accesses. Consequently, the VM resource consumption can be calculated as

$$\begin{aligned} &vmResCon_{vm_n} \\ &= \sum_{fb_x^u[r]} (repResCon_{fb_x^u[r]} \times repAccRate_{fb_x^u[r]}), \\ &\forall\ fb_x^u[r] \mid placement(fb_x^u[r]) = vm_n. \end{aligned} \tag{1}$$

The PMs' resource consumption can be calculated considering the VMs' resource consumptions and the placement relationships

$$\begin{aligned} &pmResCon_{pm_i} = \sum_{vm_n} vmResCon_{vm_n}, \\ &\forall\ vm_n \mid allocation(vm_n) = pm_i. \end{aligned} \tag{2}$$

Additionally, the hypervisor, or VM monitor (VMM), installed in the PM also consumes computational resources. This overhead is represented in our model by $hypResCon_{pm_i}$. We have assumed that this overhead is constant.

In summary, the PM resource consumption is generated by the MapReduce jobs executed in the allocated VMs and the overhead of the VMM. In any case, the model does not require a full utilization of the VM, neither of the PM.

Finally, it is also useful to define the utilization of the physical machines, pmU_{pm_i}. The utilization is a metric that measures the ratio between the consumption and the available capacity of a system resource, and its value, which ranges between 0.0 and 1.0, is calculated as

$$pmU_{pm_i} = \frac{pmResCon_{pm_i}}{pmResCap_{pm_i}}. \tag{3}$$

Three constraints emerge from the definition of the model. The first one is related to the built-in placement policy of HDFS that does not allow DataNodes to store the same replica twice.

$$\begin{aligned} &placement(fb_x^u[r]) \neq placement(fb_x^u[r']) \\ &\forall\ fb_x^u[r], fb_x^u[r'] \in fb_x^u. \end{aligned} \tag{4}$$

The second and third constraints limit the total consumption of resources in a PM, or in a VM, to be smaller than the available capacities

$$\begin{aligned} &pmResCon_{pm_i} + hypResCon_{pm_i} < pmResCap_{pm_i}, \\ &\forall\ pm_i \in PM \end{aligned} \tag{5}$$

$$vmResCon_{vm_n} < vmResCap_{vm_n}, \forall\ vm_n \in VM. \tag{6}$$

The performance metrics of a virtualized Hadoop are strongly influenced by the allocation of the VMs $allocation()$, the selection of the VM types $vmType()$, the allocation of the replicas $placement()$, the number of VMs $|VM|$, the chunk sizes $fbSize_{f_u}$, and the replication factor fR_{f_u}. These parameters are customized by the system administrator to optimize, for example, resource usage, power consumption, or data availability [7]. The remaining sections are dedicated to explaining the optimization objectives of our proposal.

3.2 Resource Waste Objective

An effective and balanced consumption of the resources facilitates the allocation or migration of the VMs [39]. An example of resource waste is a PM that has provisioned 95 percent of its main memory and 30 percent of its CPU for the already allocated VMs. Under these conditions, the allocation of new VMs will be very difficult because of the low available memory, and consequently, a high percentage of the CPU will remain unused.

Our proposed strategy is not only to allocate as many VMs as possible but also to balance the consumption of the resources. Effective and balanced use both need to be proportionally considered for all the resources.

Previous studies showed that a good indicator for the waste of resources can be formulated as [20]

$$ResourceWaste(pm_i) = \begin{cases} 0, & if \nexists\ vm_n \mid allocation(vm_n) = pm_i \\ \frac{\sigma(pmU_i)+\varepsilon}{\sum pmU_{pm_i}}, & otherwise \end{cases} \quad (7)$$

$$\sigma(pmU_{pm_i}) = \sigma(pmU^{cpu}_{pm_i}; pmU^{diskbw}_{pm_i}; pmU^{net}_{pm_i}) \quad (8)$$

$$\sum pmU_{pm_i} = pmU^{cpu}_{pm_i} + pmU^{diskbw}_{pm_i} + pmU^{net}_{pm_i}. \quad (9)$$

The resource waste is 0 when no VM is allocated since the PM could be switched off. $\sigma(pmU_{pm_i})$ is the standard deviation of the three utilizations for CPU, disk bandwidth and network bandwidth. It represents how balanced the consumption of the resources is. $\sum pmU_{pm_i}$ is the sum of those three values, and it represents the usage of the resources. Finally, the value of ε adjusts the weight of resource usage and resource balancing in the calculation. The smaller the value of the parameter is, the greater the importance that is given to the resource balance in front of the usage. We finally considered $\varepsilon = 0.15$ after the evaluation of several values.

3.3 Power Consumption Objective

The components of the PMs do not contribute to the power consumption in the same way. For example, previous studies have stated that the CPU power consumption is modeled with a piecewise linear relationship with the load, while other components present a linear consumption. We have adopted the power model presented by De Maio et al. [36]. They modeled the PM power consumption by considering the CPU, the network bandwidth and the storage bandwidth

$$PowCons(pm_i) = PowCons^{cpu}(pm_i) + PowCons^{net}(pm_i) + PowCons^{disk}(pm_i). \quad (10)$$

Their CPU power consumption model was defined as a piecewise linear relationship with the CPU load, Equation (11), and the network and storage power models were defined as linear relationships with the bandwidth usage, Equations (14) and (15). A summary of their model is [36]

$$PowCons^{cpu}(pm_i) = \begin{cases} PowCons^{cpu-low}(pm_i), & if\ pmU^{disk}_{pm_i} \le \mathcal{L}_{pm_i} \\ PowCons^{cpu-high}(pm_i), & if\ pmU^{disk}_{pm_i} \ge \mathcal{L}_{pm_i} \end{cases} \quad (11)$$

$$PowCons^{cpu-low}(pm_i) = \alpha_{pm_i} \times (pmPowMax_{pm_i} - pmPowMin_{pm_i}) \times pmU^{cpu}_{pm_i} \quad (12)$$

$$PowCons^{cpu-high}(pm_i) = \beta_{pm_i} \times (pmPowMax_{pm_i} - pmPowMin_{pm_i}) + (1 - \beta_{pm_i}) \times (pmPowMax_{pm_i} - pmPowMin_{pm_i}) \times pmU^{cpu}_{pm_i}, \quad (13)$$

where $\mathcal{L}_{pm_i}$ is the load at which the power consumption trend changes on pm_i and α_{pm_i} and β_{pm_i} are the coefficients for low and high CPU load levels, respectively.

$$PowCons^{disk}(pm_i) = \delta_{pm_i} \times (pmPowMax_{pm_i} - pmPowMin_{pm_i}) \times pmU^{disk}_{pm_i} \quad (14)$$

$$PowCons^{net}(pm_i) = \gamma_{pm_i} \times (pmPowMax_{pm_i} - pmPowMin_{pm_i}) \times pmU^{net}_{pm_i}, \quad (15)$$

where δ_{pm_i} and γ_{pm_i} are the model coefficients.

3.4 Data Unavailability Objective

HDFS stores replicas of the chunks in several DataNodes to guarantee data availability. A chunk becomes unavailable when all the nodes that store a replica are unavailable due to a temporal or permanent failure. Considering a virtualized Hadoop, a DataNode fails when either the PM or the VM fails. Thus, a replica becomes unavailable with a failure in the VM placing it or the PM allocating this VM. Data unavailability is reduced whether the VMs with the replicas of the same chunk are allocated in different PMs, and it is increased if the VMs are in the same PM. We have extended the file unavailability model presented in Long et al. [17] to the case of virtualized Hadoop

$$Unavailability(f_u) = \sum_{b^u_x \in b_u} BlockUnavailability(b^u_x). \quad (16)$$

The unavailability of a chunk, $FailureRate(b^u_x)$, is represented as

$$BlockUnavailability(b^u_x) = \prod_{pm_i \in PM(b^u_x)} \left(pmFail_{pm_i} + \prod_{vm_n \in A} vmFail_{vm_n} \right). \quad (17)$$

The outer multiplication reflects that a chunk is unavailable when all the PMs that allocate VMs storing the replicas of the chunk (b^u_x) are unavailable: $\prod_{pm_i \in PM(b^u_x)}$, where $PM(b^u_x) = \{pm_i \mid allocation(vm_n) \rightarrow pm_i\ \forall vm_n \in \{VM(b^u_x)\}$

is the set of PMs with at least one VM storing the chunk. The first term of the formula, $pmFail_{pm_i}$, represents the cases when the chunk replicas in the PM become unavailable due to a failure in that PM. The second term, the inner multiplication, represents the cases when the chunk replicas in the PM become unavailable due to failures in all the VMs that contain them: $\prod_{vm_n \in A} vmFail_{vm_n}$, where $A = VM(b_x^u) \cap \{vm_n \mid allocation(vm_n) = pm_i\}$, considering $VM(b_x^u) = \{vm_n \mid placement(b_x^u[r]) \rightarrow vm_n \ \forall b_x^u[r] \in \{b_x^u\}$ as the set of VMs containing all the replicas of a chunk.

3.5 Optimization Formulation

The objective of the optimization is to minimize file unavailability, power consumption and the waste of resources in a virtualized Hadoop. The decision variables, i.e., the elements to be managed, are the allocation of the VMs, the selection of the VM templates, the number of VM instances, and the placement of the replicas. The problem is formally defined as determining

$$|VM| \tag{18}$$

$$vmtype(vm_n) \ \forall \ vm_n \in VM \tag{19}$$

$$allocation(vm_n) \ \forall \ vm_n \in VM \tag{20}$$

$$\begin{aligned} &placement(fb_x^u[r]), \ \forall \ fb_x^u[r] \in fb_x^u, \\ &\forall \ fb_x^u \in fb_u, \ \forall \ fb_u \in HDFS, \end{aligned} \tag{21}$$

by minimizing

$$\sum_{pm_i \in PM} ResourceWaste(pm_i) \tag{22}$$

$$\sum_{pm_i \in PM} PowCons(pm_i) \tag{23}$$

$$\sum_{f_u \in HDFS} Unavailability(f_u), \tag{24}$$

subject to the definition of

$$fSize_{f_u}, \ \forall \ fb_u \in HDFS \tag{25}$$

$$fbSize_{f_u}, \ \forall \ fb_u \in HDFS, \tag{26}$$

and subject to the constraints in Equations (4), (5), and (6).

This problem has $|PM|^{|VM|}$ possible allocations of VMs, $|VM|^{|\{fb_x^u[r]\}|}$ possible placements of chunk replicas with $|VMType|$ different VM templates, and a variable and undetermined value for $|VM|$. It is an NP-hard problem since the evaluation of all the solutions is not approachable.

4 Genetic Algorithm Proposal

We propose using the non-dominated sorting genetic algorithm-II to solve our multi-objective optimization problem. Genetic algorithms (GAs) are metaheuristic approaches for solving NP-hard optimizations [40], [41], and NSGA-II is one of the most common algorithms when the optimization has multiple objectives to minimize [42]. In the field of GA and throughout this manuscript, the following terms are used indistinctly: solution, chromosome, and individual; set of solutions and population; algorithm iteration and generation; and offspring, next population, and next algorithm iteration.

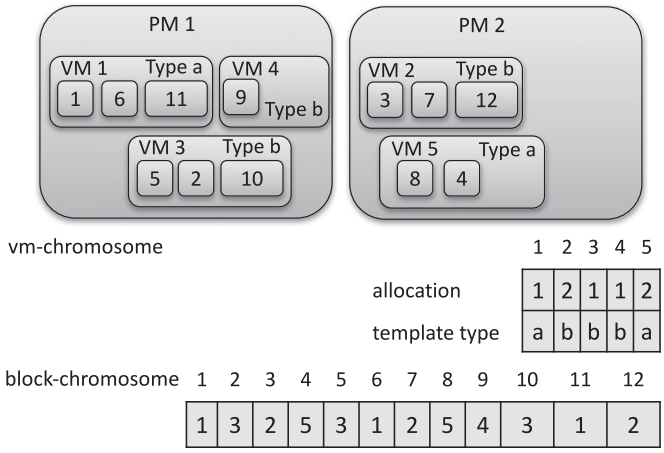


Fig. 1. Example of chromosome representation.

The implementation of a GA involves defining the chromosomes, the fitness function, the crossover and mutation operators, the selection operator, the replacement operator (offspring generation), and the execution settings. However, NSGA-II sets the selection and replacement operators, and it defines the fitness function as a vector that includes all the objective functions. The details of NSGA-II can be found in the original article [43] or in related works regarding resource management [44], [45]. The following sections explain the implementation details that are not preset by NSGA-II.

4.1 Chromosome Representation

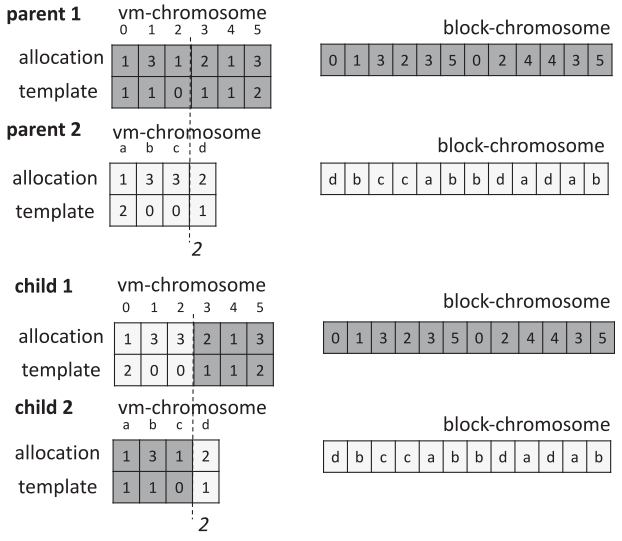
In our case, the individuals of the population represent VM allocation, VM type selection, and replica placement, considering a fixed number of replicas and a variable number of VMs. We represent each individual with two arrays: the vm-chromosome (C^{vm}) for the $allocation()$ and $vmtype()$ relationships and the block-chromosome (C^{block}) for the $placement()$ relationship. Fig. 1 shows an example with two PMs, five VMs, two VM template types, and 12 replicas.

The vm-chromosome C^{vm} is a two-dimensional array in which the column indices represent the VMs, the values of the first row (*allocation*) indicate the PM where the VM is allocated, and the values of the second row (*template type*) indicate the template for the VM. The length of the vm-chromosome can change between solutions since there will be solutions with different numbers of VMs [46].

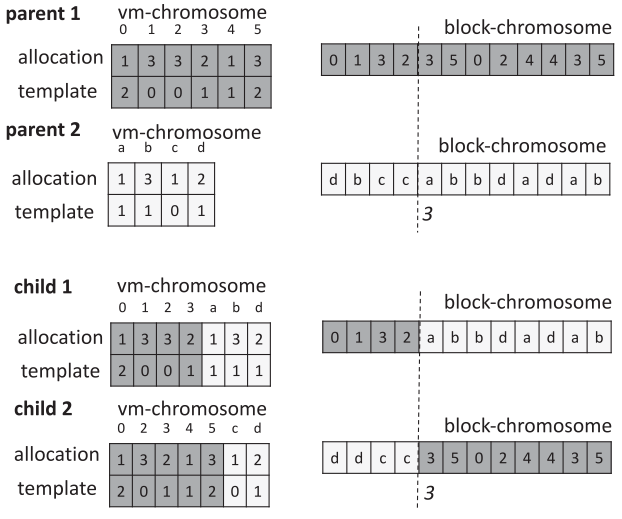
The block-chromosome is an array where the indices represent the replicas in the system ($b_u^x[r]$) and the positions contain the VM where the replica is stored. This chromosome has a fixed length since the chunk size ($fbSize_{f_u}$) and the replication factor (fR_{f_u}) are equal and constant for all the files, and it is known before the optimization process.

4.2 Crossover and Mutation Operators

GAs are based on the idea of biological evolution, where the highest qualified individuals, chosen by natural selection, are mated to obtain an offspring with the best features of both parents. In this evolution, three aspects are essential: the selection of the individuals, their mating, and random



(a) First phase: one-point crossover operator over the vm-chromosome.



(b) Second phase: one-point crossover operator over the block-chromosome.

Fig. 2. Example of the two phases of the crossover operator.

changes in the offspring. In GA, these aspects are respectively translated into selection, crossover and mutation operators.

The crossover operator defines how two parent solutions are combined to obtain two evolved children. Since we are dealing with three decision variables (vm allocation, vm template selection, and replica placement), the crossover is sequentially divided into a first phase where the crossover of the vm-chromosome is performed over the allocation and template-type arrays and a second phase for the crossover of the block-chromosome. We use one cutting-point crossover operator in both cases. This operator generates one random cutting point that splits the chromosomes into two pieces. Children's chromosomes are generated by combining the opposite pieces from both parents [47].

The one-point operator for variable chromosome length is applied to the vm-allocation [48]: one random number is generated between 0 and the minimum length of both parents' chromosomes, and it splits the allocation and template arrays into two pieces. The first child is generated by combining the left piece of one parent with the right piece of the other. Similarly, the second child is generated with the remaining pieces. Fig. 2a shows an example of this first phase of the crossover operator.

The second phase of the crossover is more complex since the combination of the block-chromosomes results in solutions with replicas allocated in VMs from both fathers. Consequently, all the VMs referenced from the new block-chromosome should be incorporated in the resulting vm-chromosome. Fig. 2b illustrates an example with the cutting point placed in position 3. In this example, the new block-chromosome of the first child results in placing all the replicas in VMs *0*, *1*, *2*, and *3* from the first parent and VMs *a*, *b*, and *d* from the second. Consequently, the vm-chromosome is generated by the concatenation of positions *0*, *1*, *2*, and *3* of the vm-chromosome from the first father (dark gray positions in the vm-chromosome) and positions *a*, *b*, and *d* from the second (light gray positions).

Formally, we define the block one cutting-point crossover similar to the splitting operation between two parents' block-chromosomes, $C_{f1} = \{C_{f1}^{vm}, C_{f1}^{block}\}$ and $C_{f2} = \{C_{f2}^{vm}, C_{f2}^{block}\}$, and one uniformly random value, j. The block-chromosomes of the new individuals are generated from the concatenation of the opposite segments of both parents split by j: $C_{c1}^{block} = C_{f1}^{block}[0..j] || C_{f2}^{block}[j+1..|C_{f2}^{block}|-1]$ and $C_{c2}^{block} = C_{f2}^{block}[0..j] || C_{f1}^{block}[j+1..|C_{f1}^{block}|-1]$. The vm-chromosomes are obtained from joining the genes of the parents corresponding to the VMs that are contained in the resulting block-chromosome. Considering $G_p^{vm}(C_c^{block}[n])$, the VM gene of the corresponding father where the block $C_c^{block}[n]$ of the corresponding child is stored, the vm-machine chromosomes of the children can be defined as $C_{c1}^{vm} = \{G_p^{vm}(C_c^{block}[n]) \ \forall \ C_c^{block}[n] \ \in C_{c1}^{block}\}$ and $C_{c2}^{vm} = \{G_p^{vm}(C_c^{block}[n]) \ \forall \ C_c^{block}[n] \ \in C_{c2}^{block}\}$.

Mutation operators generate random changes in the chromosomes of a new solution in the offspring. We define mutations for both vm and block chromosomes.

First, in the VM mutation phase, the VM growth and shrink mutations respectively increase or decrease the number of VMs with a probability of $\frac{1}{3}$. Then, the VM replace mutation iterates the positions of both vm-chromosome arrays (allocation and template) and randomly changes their values with a probability of 0.5. On average, half of the VM allocations and template types will be modified in the mutated solution.

Finally, the block replace mutation is applied to the block-chromosome by also iterating the positions of the chromosome and randomly changing their values with a probability of 0.5. The new random placements of the chunks are generated by only considering the current VMs in the solution. Therefore, the block replace mutation does not generate changes in the length of the vm-chromosome.

4.3 Genetic Algorithm Setting

The execution of a GA needs to set up some parameters that cannot be generalized between experimental domains, such as the ending condition, the generation of the initial population, and so forth. These parameters are commonly set by exploration of cases in a previous execution phase [49]. This exploratory phase consists of executing the experiment with a wide range of setting alternatives and choosing the values that obtain better results.

TABLE 2
GA and AIA Settings Defined in Preliminary Exploratory Phase

Parameter	GA	AIA
Population size	100 solutions	
Mutation probability	0.1	0.95
Suppression rate	—	0.05
Number of generations	200 iterations	
Crossover operator	Twofold one cutting-point	—
Mutation operator	VM growth, VM shrink, VM replace & block replace	Block replace
Selection operator	—	Tournament
Initial population	Round-robin with rack-awareness	
Constrain violation	Fitness takes ∞ value	

TABLE 3
Characterization of the File System

Parameter		Value
Base number of files	$\|HDFS\|$	50
Chunk size (MB)	$fbSize_{f_u}$	64
Replication factor	fR_{f_u}	3
Replica CPU cons. (%)	$replResCon^{cpu}_{fb^u_x}$	0.1
Replica disk cons. (MB/s)	$replResCon^{disk}_{fb^u_x}$	128
Replica network cons. (MB/s)	$replResCon^{net}_{fb^u_x}$	128
File sizes (MB)	$size(f_u)$	[1600, 1623, 1646, 1671, 1696, 1723, 1750, 1779, 1809, 1839, 1872, 1905, 1941, 1977, 2016, 2056, 2099, 2143, 2190, 2239, 2292, 2347,2406, 2468, 2535, 2606, 2681, 2763, 2851, 2946, 3049, 3161, 3283, 3418, 3567, 3733, 3918, 4128, 4367, 4643, 4965, 5347, 5810, 6382, 7113, 8087, 9462, 11586, 15412, 25101]
File rate (ms^{-1})	$rate(f_u)$	[128.21, 78.97, 58.46, 48.21, 41.03, 35.90, 31.79, 29.74, 26.67, 24.62, 23.59, 21.54, 20.51, 19.49, 18.46, 17.44, 17.44, 16.41, 15.38, 15.38, 14.36, 14.36, 13.33, 13.33, 13.33, 12.31, 12.31, 12.31, 11.28, 11.28, 11.28 , 10.26, 10.26, 10.26, 10.26, 10.26, 9.23, 9.23, 9.23, 9.23, 9.23, 9.23, 8.21, 8.21, 8.21, 8.21, 8.21, 8.21, 8.21, 7.18]

Source: *Xie et al. [51] and Long et al. [17].*

Table 2 summarizes the values that showed better results in the exploratory phase and that we adopted in our experiments. The main values are a population size of 100 individuals, 200 generations to finish the algorithm execution, and a mutation probability of 0.1.

In addition, the initial replica placement is performed in a round-robin distribution, but considering the rack-awareness Hadoop constraint [1], two replicas cannot be placed in the same DataNode. The allocation of the VMs is also performed in a round-robin distribution across the PMs.

The constraints need to be checked for all the solutions generated during the GA. If any of the constraints are violated, then the solution is included in the population, but its fitness value is set to infinity. Thus, the diversity of the solution space is broadened [50].

5 Experiment Design

The experiments were conducted with our own implementation of NSGA-II and a set of data structures that stored the system model (inputs) and the allocation and placement solutions (outputs). It was developed with Python 2.7, and the source code is publicly available.[1] Through the iterative execution of NSGA-II, the Pareto optimal front evolved until the finish condition was reached. The Pareto optimal front is the set of solutions that minimize our objective functions. The program calculated the values of the objective functions for each individual. The analysis of the results was performed based on those objective values.

TABLE 4
Characteristics of the PMs

Parameter	Units	pm_0	pm_1
$PowCons^{min}_{pm_i}$	Watts	501	164
$PowCons^{max}_{pm_i}$	Watts	804	382
$\mathcal{L}_{pm_i}$		0.12	0.12
α		5.29	4.33
β		0.68	0.47
δ		0.1	0.1
γ		0.05	0.05
$phyResCap^{cpu}_{pm_i}$	cores	24	12
$phyResCap^{disk}_{pm_i}$	MB/s	17800	15000
$phyResCap^{net}_{pm_i}$	MB/s	76800	38400
$fail_{pm_i}$	$week^{-1}$	[0.0015..0.019]	[0.003..0.04]

Source: *Dell [52], Birke et al. [37], and De Maio et al. [36].*

TABLE 5
Characteristics of the VMs

vm type	num. cores	disk bandwidth (MB/s)	netbandwidth (MB/s)	failurerate ($week^{-1}$)
c3.xlarge	4	250	100	[0.002..0.025]
c3.2xlarge	8	320	240	[0.004..0.05]
c3.4xlarge	16	400	200	[0.002..0.025]
m3.xlarge	4	320	100	[0.004..0.05]
m3.2xlarge	8	400	200	[0.002..0.025]

Source: *Amazon [53], and Birke et al. [37].*

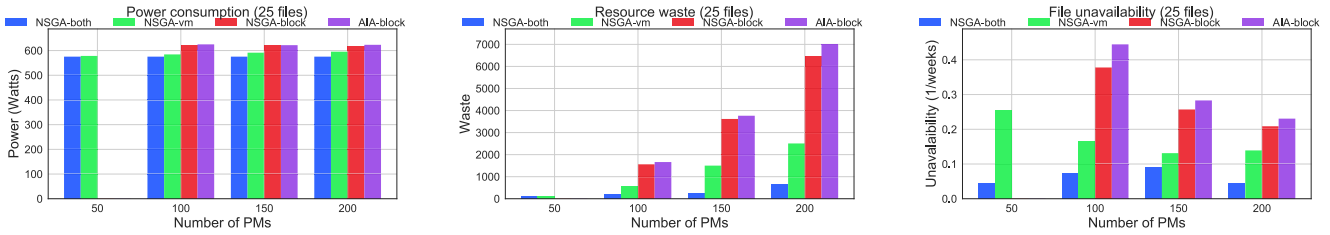
The experiments were characterized with a base definition, which was subsequently varied to study different experiment sizes. These experiment variations were defined by sequentially increasing the number of PMs—for values 50, 100, 150, and 200—, and the number of files—for values 25, 50, 100, and 200—, resulting in experiments with a total number of replicas between 2,286 and 37,200.

The model parameters for the base experiment were defined using the settings of previous works related with cloud, VMs, or HDFS. Tables 3, 4, and 5 include the values for the model parameters and the sources from which these values were obtained.

File sizes and access rates for 50 files were defined from [51]. Experiments with more files sequentially repeated these characteristics for the files after the first 50. Two PM templates were considered, and each half of the PMs was assigned to one of these templates. The VM templates were also uniformly assigned to the VM instances.

The validation of our approach was performed through comparisons with the following studies: (i) Long et al. [17] proposed improving the optimization of the HDFS placement using an artificial immune algorithm (AIA); (ii) Adamuthe et al. [18] proposed optimizing the VM allocation

1. Available at http://github.com/acsicuib/NSGA2VmHdfs/



(a) Optimized values for power consumption. (b) Optimized values for resource waste. (c) Optimized values for file unavailability.

Fig. 3. Objective values for our proposal *NSGA-both*; Adamuthe [18] *NSGA-vm*; Long [17] *AIA-block*; and *NSGA-block*. Experiments with 25 files.

using NSGA-II. We considered four experimental scenarios that were determined by the optimization algorithm and the scope: *NSGA-both*, our current proposal based on using NSGA-II simultaneously in the replica placement and in the VM selection and allocation; *NSGA-vm*, the proposal based on the work of Adamuthe et al. [18], where NSGA-II is used to optimize the VM allocation; *AIA-block*, the proposal based on MORM [17], which is an AIA-based optimization for replica placement; and *NSGA-block*, which is replica placement optimization with NSGA-II, defined with a completeness purpose.

The differences in terms of scope among the implementations of the experimental scenarios are basically in the crossover and mutation operators. *NSGA-both* implements the operators as we explained in Section 4.2. *NSGA-vm* only applies the first phase of the crossover, corresponding to the vm-chromosome, and the three mutations for this chromosome (vm growth, shrink, and replace mutations). *NSGA-block* only considers the second phase of the crossover, the one of the block-chromosome, and its unique mutation (block replace mutation). Finally, *AIA-block* does not consider any crossover since an AIA only applies mutations and the replica replace mutation in our particular case. The details of the implementation of AIA can be found in the work of Long et al. [17], and the execution settings that we used can be found in Table 2. The values for the case of the GA and AIA were fixed to be as equal as possible. However, some of the settings are not specific for NSGA-II, e.g., suppression rate, and others are not comparable, e.g., mutation probability. In those cases, we conducted a preliminary exploration phase to find the most suitable values, as in the case of NSGA-II [49].

6 Results and Discussion

The output of a multi-objective optimization is a Pareto optimal front. This is a set that includes the solutions that are not dominated by any other. A non-dominated solution has at least one objective with a smaller value than all the other solutions [43].

The analysis of the results is performed by first comparing one solution from the Pareto sets of each experimental scenario to later compare with the entire Pareto optimal sets. The former comparison requires selecting one solution from each Pareto front. There are many studies in the literature related to decision-making methodologies for selecting the preferred solution from the Pareto front [54]. We chose a straightforward method since the solution selection process was out of the scope of the comparison between the results of the optimization algorithms. We used the weighted sum of the objectives [55], setting the same weight for the three objectives since we considered the same importance for all of them. Thus, a normalized uniformly weighted sum was applied to the multi-objective vector, Equation (27), and the solution with the smallest value was selected.

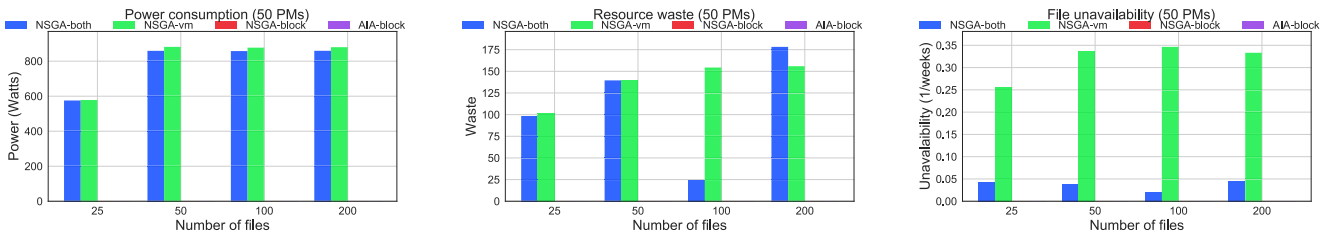
$$ws^a = \frac{rw^a - rw^{min}}{(rw^{max} - rw^{min}) \times 3} + \frac{pc^a - pc^{min}}{(pc^{max} - pc^{min}) \times 3} + \frac{fu^a - fu^{min}}{(fu^{max} - fu^{min}) \times 3}, \quad (27)$$

where rw^a, pc^a, and fu^a are the objective values for a solution a, and rw^{min}, rw^{max}, pc^{min}, pc^{max}, fu^{min}, and fu^{max} are each objectives' minimal and maximum values in the Pareto front.

Figs. 3, 4, 5, and 6 show the comparison of the optimized objectives for the selected solutions between each experimental scenario. As representative cases, the figures respectively show the results for the experiments with sizes of 25 files, 50 PMs, 200 files, and 200 PMs, i.e., the cases with the lowest and greatest numbers of files and PMs.

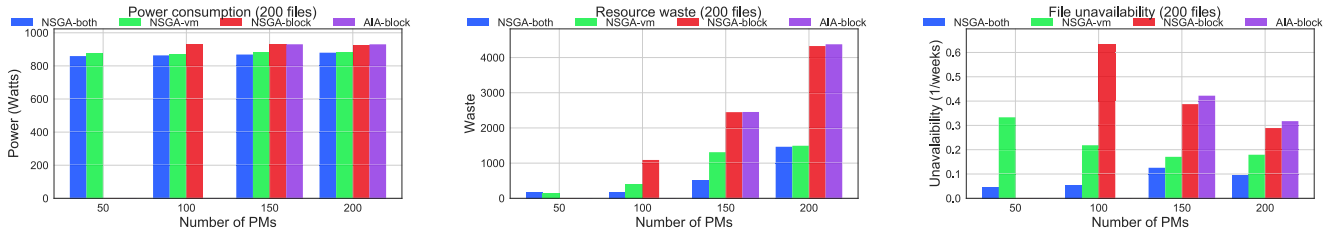
Our approach obtained better optimizations than the other scenarios, i.e., the objective values were smaller. The second best algorithm was *NSGA-vm*. *NSGA-block* and *AIA-block* obtained the worst objective values. Although the figures show only a subset of the results, the same patterns were observed in all the experimental configurations.

NSGA-block and *AIA-block* did not gather optimized solutions for the experimental configurations with 50 PMs (all



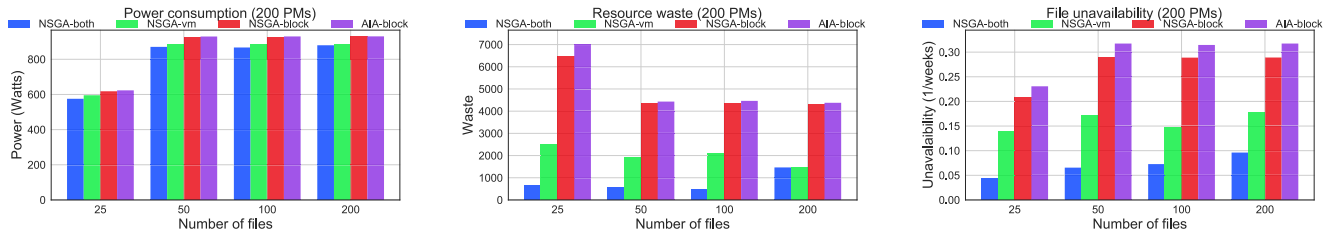
(a) Optimized values for power consumption. (b) Optimized values for resource waste. (c) Optimized values for file unavailability.

Fig. 4. Objective values for our proposal *NSGA-both*; Adamuthe [18] *NSGA-vm*; Long [17] *AIA-block*; and *NSGA-block*. Experiments with 50 PMs.



(a) Optimized values for power consumption. (b) Optimized values for resource waste. (c) Optimized values for file unavailability.

Fig. 5. Objective values for our proposal *NSGA-both*; Adamuthe [18] *NSGA-vm*; Long [17] *AIA-block*; and *NSGA-block*. Experiments with 200 files.



(a) Optimized values for power consumption. (b) Optimized values for resource waste. (c) Optimized values for file unavailability.

Fig. 6. Objective values for our proposal *NSGA-both*; Adamuthe [18] *NSGA-vm*; Long [17] *AIA-block*; and *NSGA-block*. Experiments with 200 PMs.

the cases within Fig. 4 and the first ticks within Figs. 3 and 5). All the solutions throughout and after 200 generations violated the resource usage constraints, Equations (5) and (6). In these experimental configurations, the process to find a solution that satisfied the constraints was more complicated since they had the lowest resources with only 50 PMs, and consequently, the free resources after the placement process were also very low. The results reflected that policies only based on the replica placement, as *NSGA-block* and *AIA-block*, limited the process of evolving the solution population and consequently also limited the exploration of the solution space compared to policies that combined replica and VM management. This result was even more pronounced for the case of *AIA-block*, which did not gather suitable solutions in the case of 100 PMs and 200 files. This algorithm applied only mutations and did not apply crossovers, thus making its limited flexibility to increase the solution space more clear.

With the purpose of showing the results for all the experiment sizes, we have included Fig. 7. For a clearer analysis, this figure only includes the results of our proposal and *NSGA-vm*, the best of the other three.[2] The experiments are labeled on the x-axis with the number of PMs (p) and the number of files (f), e.g., *100p-50f* corresponds to the experiment with a size of 100 PMs and 50 files. The bar charts plot the optimized values. The improvement percentages of our solution with respect to *NSGA-vm* are also included in the figure with line charts and their values. Positive improvement percentages indicate that our algorithm obtained better optimizations. In contrast, negative values correspond to cases where it obtained worse objective values than *NSGA-vm*. All the cases had positive values, except for the resource waste in the cases with 50 PMs-200 files and 150 PMs-100 files.

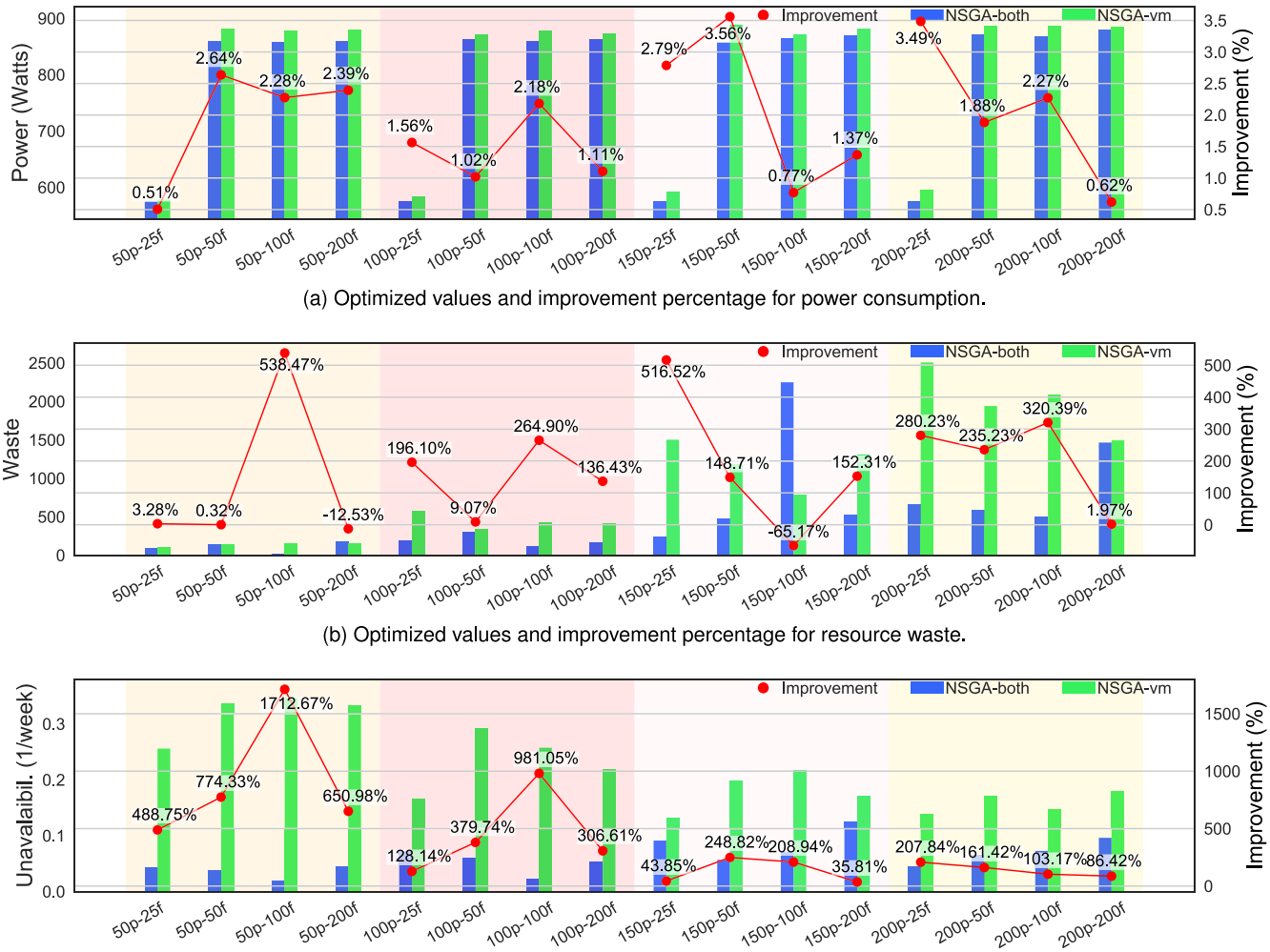
Our approach obtained the highest benefits in terms of file unavailability. It obtained 407.41 percent more availability than *NSGA-vm* on average. This improvement percentage ranged between 35.81 and 1712.67 percent. Nevertheless, the power consumption was the objective with the smallest improvements since it only saved 1.9 percent more power on average, ranging between 0.51 and 3.56 percent. Finally, our solution wasted 170.39 percent less resources than *NSGA-vm* on average. However, it showed extreme cases, from improvements of 538.47 percent to two cases with decreases of -65.17 and -12.53 percent.

File unavailability depends on the replica placement and on the VM allocation, Equation (17). Policies that only manage VM migrations are less flexible, and their effects are only reflected in one component of the unavailability model. Consider an example of two PMs (pm_1 and pm_2), three VMs (vm_1, vm_2, and vm_3), three chunks (fb_a^1, fb_b^1, and fb_c^1), and a replica placement such as $vm_1 = (fb_a^1[1], fb_b^1[2])$, $vm_2 = (fb_b^1[1], fb_c^1[2])$, and $vm_2 = (fb_c^1[1], fb_a^1[2])$. Under these conditions, all the possible VM allocations place the replicas of at least one chunk in the same PM, reducing the file unavailability. In this example, only a new replica placement could improve the unavailability. This illustrative example explains the benefits of a twofold management policy in terms of file unavailability.

Resource waste depends on PM usage and consequently on the VM allocation and the resource consumptions of the VMs, Equations (2), (3) and (7). The VM consumption depends on the replica allocation and the task resource consumptions and execution rates, Equation (1). As in the case of file unavailability, a policy that only manages the VMs is less flexible than when both VM and replica placements are managed. Suppose that we have one VM with high CPU and low disk consumption. This VM generates high waste of resources regardless of the PM that allocates it. VM migrations are not sufficient to balance the resource usage. It is necessary to redistribute the replica placement to achieve VMs with more balanced conditions. This again explains the benefits of our approach in front of *NSGA-vm*.

These results validate our initial hypothesis that better optimizations are obtained with a simultaneous and coordinated management in our study domain.

2. The complete set of results, also including the other two scenarios, is available in folder *article_results* in the source code repository.



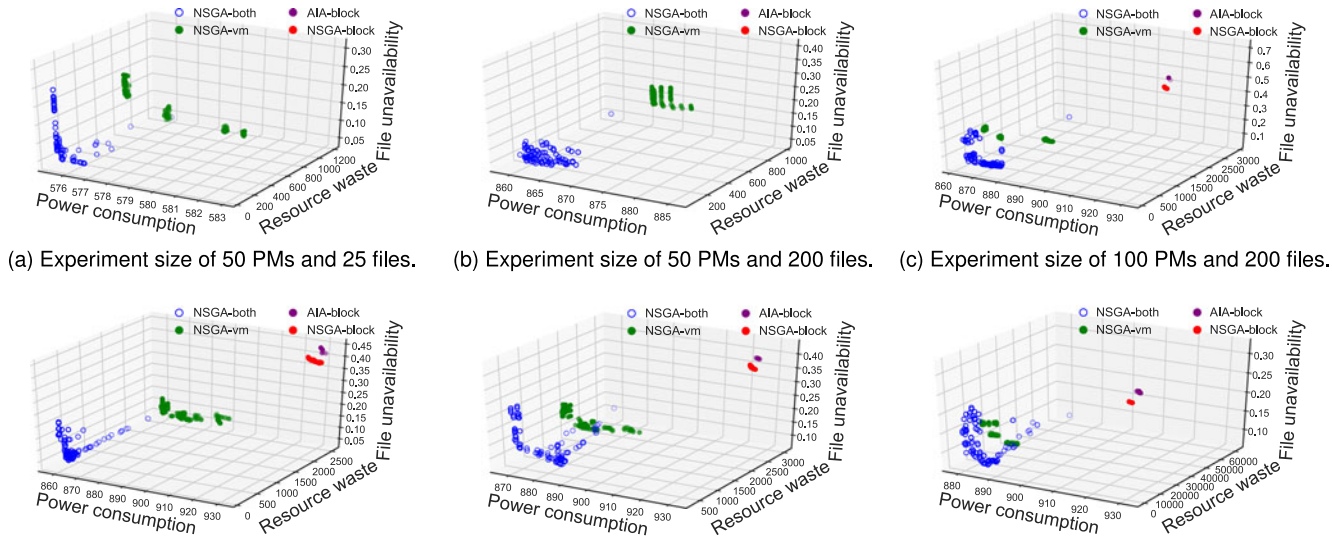
(a) Optimized values and improvement percentage for power consumption.

(b) Optimized values and improvement percentage for resource waste.

(c) Optimized values and improvement percentage for file unavailability.

Fig. 7. Objective comparison and improvement percentage of our approach (*NSGA-both*) with respect to Adamuthe [18] (*NSGA-vm*).

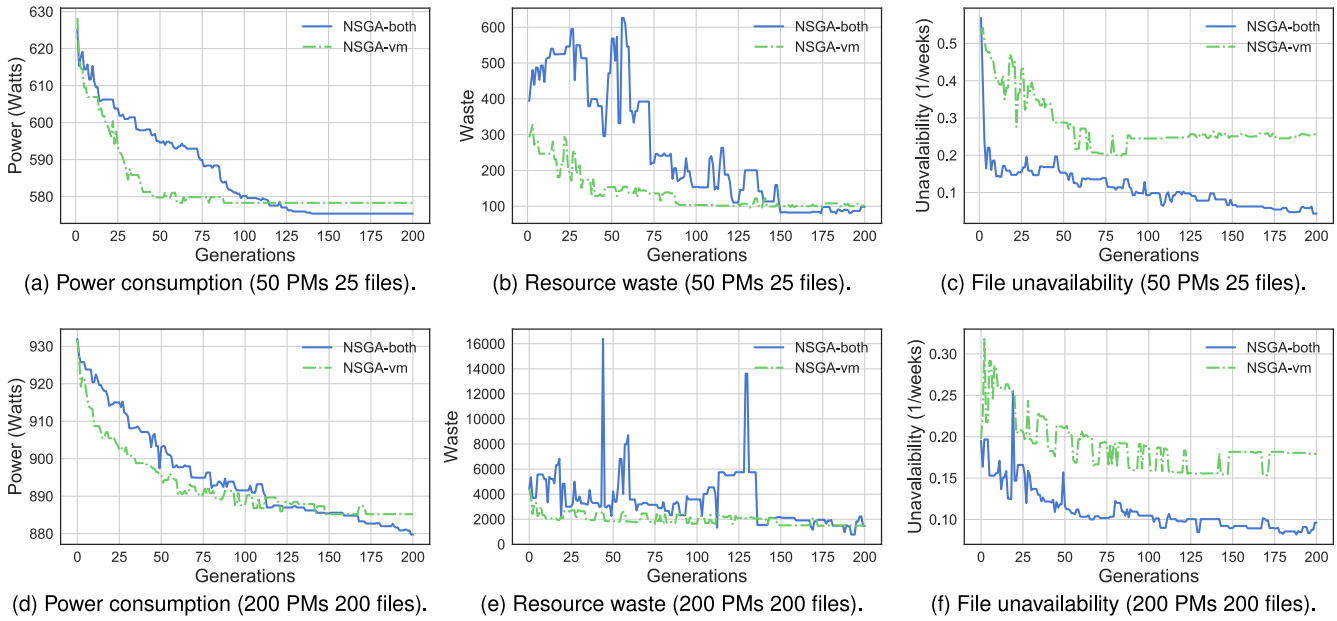
Fig. 8 shows the scatter plots of the solutions within the Pareto optimal front for some sizes of the experiments. Each point in the plots corresponds to a solution in the Pareto front, and it represents the values of the three objectives. The more the solutions are plotted to the left, bottom, or front positions, the better they are. The comparison of the Pareto optimal front allowed us to draw conclusions about all the solutions without being conditioned by the method



(a) Experiment size of 50 PMs and 25 files. (b) Experiment size of 50 PMs and 200 files. (c) Experiment size of 100 PMs and 200 files.

(d) Experiment size of 150 PMs and 50 files. (e) Experiment size of 150 PMs and 200 files. (f) Experiment size of 200 PMs and 200 files.

Fig. 8. Pareto optimal front comparison for several sizes of the experiments.



(a) Power consumption (50 PMs 25 files). (b) Resource waste (50 PMs 25 files). (c) File unavailability (50 PMs 25 files).

(d) Power consumption (200 PMs 200 files). (e) Resource waste (200 PMs 200 files). (f) File unavailability (200 PMs 200 files).

Fig. 9. Evolution of the objective values for the experiments with 50 PMs / 25 files and 200 PMs / 200 files.

for selecting one of them. For all the cases, the Pareto optimal fronts obtained with our proposal are closer to the coordinate origin than the other Pareto fronts.

The scatter plots of the Pareto sets also permit the analysis of the diversity in the solution space. A greater dispersion of the optimized solutions is commonly considered as a higher quality optimization since final users have more flexibility if their requirements or preferences to select the preferred solution change. The Pareto fronts obtained with our algorithm showed higher diversity for the solution space than the other alternatives. At the other end, *NSGA-block* and *AIA-block* showed a high concentration of the solutions, reducing the alternatives to select one solution.

To summarize, we conclude that the management of the allocation of VMs achieves higher optimizations than the management of the replica placement in our particular domain. However, the improvements are higher when the management of both elements is performed simultaneously and in coordination.

7 Genetic Algorithm Deployment

7.1 Performance Analysis

We analyzed the execution performance of the algorithm in terms of the stabilization of the objectives and the execution time.

Fig. 9 shows the evolution of the objectives along the generations of the GA execution. To provide a clear analysis, we only present the results for our approach and for *NSGA-vm*. Our approach achieved the smallest values for the objectives around generation number 150 in the worst cases. In the case of the *NSGA-vm* algorithm, it stabilized the objective values around generation 100. The figure only shows the cases for the smallest and the largest experiments, but the same pattern was observed for all sizes of experiments.

The experiments were executed with Python 2.7.6 on an Ubuntu Server 14.04.5 LTS distribution installed on an Intel (R) Core(TM) i7-4770S 3.10 GHz CPU with 8 GB of RAM. The execution times are measured separately for the optimization process and the model calculations. Table 6 shows the execution times for the smallest and largest experiment sizes: the column *Population* presents the time for crossover and, in the cases of NSGA-II, also the mutation operators; the column *Selection* sums the times for crowding distance, ordering, and, only in the cases of NSGA-II, the front generation; the column *Usage* presents the PMs and VMs usage calculation times; and the column *Fitness* presents the fitness calculations and constraint checking times. By considering a number of 150 generations to optimize the solutions in the case of our approach, the executions time ranged between 80 and 950 s compared with 46 and 569 s in the case of, for example, *NSGA-vm* and considering 100 generations.

Note that approximately 90-96 percent of the execution time was taken by model calculations and only 4-10 percent in the optimization process. Moreover, approximately 70 percent of the total execution time was taken by the file unavailability objective calculation, resulting in it being the most time-demanding task. Furthermore, the execution time of model calculations is equal, regardless of the optimization algorithm used in the proposed solution. In general terms, the model calculation times were homogeneous among the four experimental scenarios. The differences appeared in the optimization process, where AIA showed lowest values. However, this benefit was not important since it corresponded to the phase with the smallest percentage in the total execution time.

TABLE 6
Time Taken in Milliseconds to Obtain a New Generation in the GA

Experiment	Optimization		Model	
	Population min/max	Selection min/max	Usage min/max	Fitness min/max
NSGA-both	7/617	14/15	123/1449	395/4311
NSGA-vm	7/93	15/15	84/1421	363/4169
NSGA-block	14/251	13/15	82/1342	312/4182
AIA-block	1/18	3/5	82/1397	339/4399

Although the execution times of evolutionary strategies are generally greater than those of other alternatives, they are still common solutions for resource management in cloud architectures [16], [56]. These optimization tools are commonly applied in static or offline approaches, such as VM placement, service placement or workflow scheduling [56]. Note that virtualized Hadoop platforms are very dynamic scenarios, but replica and VM migration are time-consuming operations that cannot be performed continuously. Therefore, evolutionary approaches can be applied periodically, with a time slot larger than the optimization execution time, as it is commonly done in other previous studies [57]. The execution time can also be reduced by limiting the optimization to a subset of elements in the system, such as the most popular files or the files that generate higher loads in the system [32].

An alternative way to reduce execution times is to take advantage of the nature of the architecture and to implement a MapReduce version of NSGA-II [58]. A suitable approach would be to calculate the fitness functions in the map phase, the random selection of the individual in the shuffle phase, and finally, the crossover and mutation operators in the reduce phase. In this way, the highest time-consuming calculations (usage and fitness) would be parallelized with the map tasks.

Finally, we also measured the computational complexity considering P number of PMs, V number of VMs, R number of replicas, S population size, M number of objectives, and F replication factor. Resource waste and energy consumption are calculated with only one iteration over the PMs: $O(P)$. Unavailability analyzes the replicas of each chunk several times, $O(V + P + RF + 2F^2)$. Usage constraints are calculated with one iteration over the set of PMs and over the set of VMs: $O(P) + O(V)$. The rack-awareness constraint needs one iteration over the set of replicas: $O(R)$. Crossover requires traveling the block-chromosome to fix unmet dependencies between replicas and VMs: $O(R)$. Finally, the mutation operator iterates over both chromosomes, vm and block, $O(V + R)$. The overall computational complexity considering the previous tasks for each solution is $O(3VS + 3PS + 3RS + RFS + 2SF^2)$. Finally, the overall complexity of NSGA-II is measured as $O(MS^2)$ [43].

7.2 Real Scenario Integration

There are several alternatives for implementing our approach in a real infrastructure. We propose implementing it as a plug-in whose inputs are metrics from the system and the output is the actions to be taken by Hadoop and the hypervisor or VM monitor.

In the first step, the inputs would be gathered from several components: from the NaneNode, the file number, chunk number, or replication factor; from the JobTracker, the job execution rates; from the VMM the VM number, types, virtual resources, or failures metrics; and from the hardware features of the PMs, the PM number, physical resources, failure metrics, or power consumptions features. The gathering of these data could be implemented by monitoring daemons using the Hadoop and the VMM daemon or even with other existing monitoring tools, such as Ambari for Hadoop and Nagios for VMs [59].

Our optimization process could be run periodically or according to preconfigured events. Both triggers would be controlled by these monitoring tools. Each run would return the replica placement (block-chromosome) and the VM allocation (vm-chromosome). The replica placements would be integrated into the NaneNode using the Hadoop API, and VM allocations would be conducted according to the API of the selected VMM, e.g., VMware API.

8 Conclusion

We presented a simultaneous management of VM allocation, VM template selection and replica placement in virtualized Hadoop. Our solution minimizes three objectives: power consumption of the physical machines, waste of physical resources, and file unavailability. We have implemented the solution with NSGA-II, using a twofold chromosome and crossover operator to address this multi-objective optimization problem. Our solution has been compared with the proposals from Adamuthe [18] and Long [17], and it obtained important improvements.

The benefits in power consumption were measured to have an overall improvement of 1.9 percent. It showed a better behavior with the resource waste and file unavailability, obtaining average improvements of 170.38 and 407.41 percent, respectively. These benefits were achieved at the expense of a larger number of generations and longer execution times than the second best scenario, the one that only managed the VMs.

Our results open new challenges and we consider three ongoing research lines. The first is to adapt the decision variables to study the benefits of a variable replication factor and chunk size. Our second active research line is to add an additional objective to obtain solutions that also reduce the cost of live migrations of VMs and file replicas. Finally, we also plan to study the applicability of simultaneous management in other cloud/distributed architectures, such as microservice-based or fog domains.

Acknowledgments

This research was supported by the Spanish Government (Agencia Estatal de Investigación) and the European Commission (Fondo Europeo de Desarrollo Regional) through grant number TIN2017-88547-P (MINECO/AEI/FEDER, UE).

References

[1] D. Borthakur, "HDFS architecture guide," Hadoop Project Website. (2008). [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.pdf

[2] T. Ivanov, R. V. Zicari, S. Izberovic, and K. Tolle, "Performance evaluation of virtualized hadoop clusters," *CoRR*, 2014. [Online]. Available: http://arxiv.org/abs/1411.3811

[3] A. Raj, K. Kaur, U. Dutta, V. V. Sandeep, and S. Rao, "Enhancement of hadoop clusters with virtualization using the capacity scheduler," in *Proc. 3rd Int. Conf. Serv. Emerging Markets*, Dec. 2012, pp. 50–57.

[4] A. Group, "Virtual hadoop," Apache Hadoop Wiki. (2017). [Online]. Available: https://wiki.apache.org/hadoop/ 927 Virtual %20Hadoop

[5] Z. A. Mann, "Allocation of virtual machines in cloud data centers-A survey of problem models and optimization algorithms," *ACM Comput. Surv.*, vol. 48, no. 1, pp. 11:1–11:34, Aug. 2015. [Online]. Available: http://doi.acm.org/10.1145/2797211

[6] K. A. Kumar, A. Quamar, A. Deshpande, and S. Khuller, "SWORD: Workload-aware data placement and replica selection for cloud data management systems," *VLDB J.*, vol. 23, no. 6, pp. 845–870, Dec. 2014. [Online]. Available: https://doi.org/10.1007/s00778–014-0362-1

[7] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: State-of-the-art and research challenges," *J. Internet Serv. Appl.*, vol. 1, no. 1, pp. 7–18, 2010. [Online]. Available: http://dx.doi.org/10.1007/s13174–010-0007-6

[8] S. Singh and I. Chana, "Cloud resource provisioning: Survey, status and future research directions," *Knowl. Inf. Syst.*, vol. 49, no. 3, pp. 1005–1069, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10115–016-0922-3

[9] R. K. Grace and R. Manimegalai, "Dynamic replica placement and selection strategies in data grids. A comprehensive survey," *J. Parallel Distrib. Comput.*, vol. 74, no. 2, pp. 2099–2108, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731513002207

[10] T. Hamrouni, S. Slimani, and F. B. Charrada, "A survey of dynamic replication and replica selection strategies based on data mining techniques in data grids," *Eng. Appl. Artif. Intell.*, vol. 48, pp. 140–158, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0952197615002493

[11] S. U. R. Malik, S. U. Khan, S. J. Ewen, N. Tziritas, J. Kolodziej, A. Y. Zomaya, S. A. Madani, N. Min-Allah, L. Wang, C.-Z. Xu, Q. M. Malluhi, J. E. Pecero, P. Balaji, A. Vishnu, R. Ranjan, S. Zeadally, and H. Li, "Performance analysis of data intensive cloud systems based on data management and replication: A survey," *Distrib. Parallel Databases*, vol. 34, no. 2, pp. 179–215, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10619–015-7173-2

[12] B. A. Milani and N. J. Navimipour, "A comprehensive review of the data replication techniques in the cloud environments: Major trends and future directions," *J. Netw. Comput. Appl.*, vol. 64, pp. 229–238, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804516000795

[13] A. K. Karun and K. Chitharanjan, "A review on hadoop: HDFS infrastructure extensions," in *Proc. IEEE Conf. Inf. Commun. Technol.*, Apr. 2013, pp. 132–137.

[14] C. Guerrero, I. Lera, and C. Juiz, "Migration-aware genetic optimization for MapReduce scheduling and replica placement in hadoop," *J. Grid Comput.*, vol. 16, no. 2, pp. 265–284, Jun. 2018. [Online]. Available: https://doi.org/10.1007/s10723-018-9432-8

[15] S. Singh and I. Chana, "A survey on resource scheduling in cloud computing: Issues and challenges," *J. Grid Comput.*, vol. 14, no. 2, pp. 217–264, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10723–015-9359-2

[16] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Comput. Surv.*, vol. 47, no. 4, pp. 63:1–63:33, Jul. 2015. [Online]. Available: http://doi.acm.org/10.1145/2788397

[17] S.-Q. Long, Y.-L. Zhao, and W. Chen, "MORM: A multi-objective optimized replication management strategy for cloud storage cluster," *J. Syst. Archit.*, vol. 60, no. 2, pp. 234–244, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762113002671

[18] A. C. Adamuthe, R. M. Pandharpatte, and G. T. Thampi, "Multiobjective virtual machine placement in cloud environment," in *Proc. Int. Conf. Cloud Ubiquitous Comput. Emerging Technol.*, 2013, pp. 8–13. [Online]. Available: http://dx.doi.org/10.1109/CUBE.2013.12

[19] Z. Zhu, G. Zhang, M. Li, and X. Liu, "Evolutionary multi-objective workflow scheduling in cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1344–1357, May 2016.

[20] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu, "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing," *J. Comput. Syst. Sci.*, vol. 79, no. 8, pp. 1230–1242, Dec. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.jcss.2013.02.004

[21] S. Su, Y. Su, F. Shao, and H. Guo, "A power-aware virtual machine mapper using firefly optimization," in *Proc. 3rd Int. Conf. Advanced Cloud Big Data*, 2015, pp. 96–103. [Online]. Available: http://dx.doi.org/10.1109/CBD.2015.25

[22] Y. Kessaci, N. Melab, and E.-G. Talbi, "An energy-aware multi-start local search heuristic for scheduling VMs on the OpenNebula cloud distribution," in *Proc. Int. Conf. High Perform. Comput. Simul.*, 2012, pp. 112–118. [Online]. Available: http://dblp.uni-trier.de/db/conf/ieeehpcs/ieeehpcs2012.html#KessaciMT12

[23] F. López-Pires and B. Barán, "Many-objective virtual machine placement," *J. Grid Comput.*, vol. 15, no. 2, pp. 161–176, Jun. 2017. [Online]. Available: https://doi.org/10.1007/s10723–017-9399-x

[24] J. Xu and J. A. B. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *Proc. IEEE/ACM Int Conf. Green Comput. Commun. Int. Conf. Cyber Phys. Social Comput.*, 2010, pp. 179–188. [Online]. Available: http://dx.doi.org/10.1109/GreenCom-CPSCom.2010.137

[25] H. Mi, H. Wang, G. Yin, Y. Zhou, D. Shi, and L. Yuan, "Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers," in *Proc. IEEE Int. Conf. Serv. Comput.*, Jul. 2010, pp. 514–521.

[26] P. Basanta-Val, N. C. Audsley, A. J. Wellings, I. Gray, and N. Fernández-García, "Architecting time-critical big-data systems," *IEEE Trans. Big Data*, vol. 2, no. 4, pp. 310–324, Dec. 2016.

[27] P. Basanta-Val, N. Fernández-García, L. Sánchez-Fernández, and J. Arias-Fisteus, "Patterns for distributed real-time stream processing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 11, pp. 3243–3257, Nov. 2017.

[28] J. Song, H. He, Z. Wang, G. Yu, and J.-M. Pierson, "Modulo based data placement algorithm for energy consumption optimization of MapReduce system," *J. Grid Comput.*, pp. 1–16, 2016. [Online]. Available: https://doi.org/10.1007/s10723-016-9370-2

[29] W. Dai, I. Ibrahim, and M. Bassiouni, "A new replica placement policy for hadoop distributed file system," in *Proc. IEEE 2nd Int. Conf. Big Data Secur. Cloud*, Apr. 2016, pp. 262–267.

[30] M. Y. Eltabakh, Y. Tian, F. Özcan, R. Gemulla, A. Krettek, and J. McPherson, "CoHadoop: Flexible data placement and its exploitation in hadoop," *Proc. VLDB Endowment*, vol. 4, no. 9, pp. 575–585, Jun. 2011. [Online]. Available: http://dx.doi.org/10.14778/2002938.2002943

[31] N. Maheshwari, R. Nanduri, and V. Varma, "Dynamic energy efficient data placement and cluster reconfiguration algorithm for MapReduce framework," *Future Generation Comput. Syst.*, vol. 28, no. 1, pp. 119–127, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X1100135X

[32] Z. Cheng, Z. Luan, Y. Meng, Y. Xu, D. Qian, A. Roy, N. Zhang, and G. Guan, "ERMS: An elastic replication management system for HDFS," in *Proc. IEEE Int. Conf. Cluster Comput. Workshops*, Sep. 2012, pp. 32–40.

[33] Q. Wei, B. Veeravalli, B. Gong, L. Zeng, and D. Feng, "CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster," in *Proc. IEEE Int. Conf. Cluster Comput.*, Sept 2010, pp. 188–196.

[34] X. Wang, Y. Wang, and Y. Cui, "A new multi-objective bi-level programming model for energy and locality aware multi-job scheduling in cloud computing," *Future Generation Comput. Syst.*, vol. 36, pp. 91–101, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X13002689

[35] L. Lu, X. Shi, H. Jin, Q. Wang, D. Yuan, and S. Wu, "Morpho: A decoupled MapReduce framework for elastic cloud computing," *Future Generation Comput. Syst.*, vol. 36, pp. 80–90, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X13002902

[36] V. D. Maio, G. Kecskemeti, and R. Prodan, "An improved model for live migration in data centre simulators," in *Proc. 16th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, May 2016, pp. 527–530.

[37] R. Birke, I. Giurgiu, L. Y. Chen, D. Wiesmann, and T. Engbersen, "Failure analysis of virtual and physical machines: Patterns, causes and characteristics," in *Proc. 44th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw.*, Jun. 2014, pp. 1–12.

[38] A. Sheth, "A new landscape for distributed and parallel data management," *Distrib. Parallel Databases*, vol. 30, no. 2, pp. 101–103, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10619–012-7091-5

[39] K. S. Rao and P. S. Thilagam, "Heuristics based server consolidation with residual resource defragmentation in cloud data centers," *Future Generation Comput. Syst.*, vol. 50, pp. 87–98, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X14001794

[40] H. Li and Q. Zhang, "Multiobjective optimization problems with complicated pareto sets, MOEA/D and NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 284–302, Apr. 2009.

[41] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Rel. Eng. Syst. Safety*, vol. 91, no. 9, pp. 992–1007, 2006.

[42] C. von Lücken, B. Barán, and C. Brizuela, "A survey on multi-objective evolutionary algorithms for many-objective problems," *Comput. Optimization Appl.*, vol. 58, no. 3, pp. 707–756, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10589–014-9644-1
[43] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002. [Online]. Available: http://dx.doi.org/10.1109/4235.996017
[44] C. Guerrero, I. Lera, and C. Juiz, "Genetic algorithm for multi-objective optimization of container allocation in cloud architecture," *J. Grid Comput.*, vol. 16, no. 1, pp. 113–135, Mar. 2018. [Online]. Available: https://doi.org/10.1007/s10723–017-9419-x
[45] C. Guerrero, I. Lera, and C. Juiz, "Resource optimization of container orchestration: A case study in multi-cloud microservices-based applications," *J. Supercomput.*, pp. 1–28, Apr. 2018. [Online]. Available: https://doi.org/10.1007/s11227-018-2345-2
[46] C. Y. Lee, "Variable length genomes for evolutionary algorithms," in *Proc. Genetic Evol. Comput. Conf.*, 2000, Art. no. 806.
[47] K. Deb, "Multi-objective genetic algorithms: Problem difficulties and construction of test problems," *Evol. Comput.*, vol. 7, pp. 205–230, 1999.
[48] A. H. Brie and P. Morignot, "Genetic planning using variable length chromosomes," in *Proc. 15th Int. Conf. Int. Conf. Automated Planning Scheduling*, 2005, pp. 320–329. [Online]. Available: http://dblp.uni-trier.de/db/conf/aips/icaps2005.html#BrieM05
[49] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Trans. Syst. Man Cybern.*, vol. 16, no. 1, pp. 122–128, Jan. 1986.
[50] Z. Michalewicz, "A survey of constraint handling techniques in evolutionary computation methods," *Evol. Program.*, vol. 4, pp. 135–155, 1995.
[51] T. Xie and Y. Sun, "A file assignment strategy independent of workload characteristic assumptions," *Trans. Storage*, vol. 5, no. 3, pp. 10:1–10:24, Nov. 2009. [Online]. Available: http://doi.acm.org/10.1145/1629075.1629079
[52] Dell, "Enterprise infrastructure planning tool," Dell, Round Rock, TX, Tech. Rep., 2016. [Online]. Available: http://www.dell.com/calc
[53] A. W. Services, "Amazon EC2 instance types," Amazon., Seattle, WA, Tech. Rep., 2016. [Online]. Available: https://aws.amazon.com/ec2/instance-types/
[54] E. Zio and R. Bazzo, *A Comparison of Methods for Selecting Preferred Solutions in Multiobjective Decision Making*. Paris, France: Atlantis Press, 2012, pp. 23–43. [Online]. Available: https://doi.org/10.2991/978–94-91216-77-0_2
[55] R. T. Marler and J. S. Arora, "The weighted sum method for multi-objective optimization: New insights," *Struct. Multidisciplinary Optimization*, vol. 41, no. 6, pp. 853–862, Jun. 2010. [Online]. Available: https://doi.org/10.1007/s00158–009-0460-7
[56] M. Guzek, P. Bouvry, and E. G. Talbi, "A survey of evolutionary computation for resource management of processing in cloud computing [review article]," *IEEE Comput. Intell. Mag.*, vol. 10, no. 2, pp. 53–67, May 2015.
[57] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, "Optimized IoT service placement in the fog," *Serv. Oriented Comput. Appl.*, vol. 11, pp. 427–443, Oct. 2017. [Online]. Available: https://doi.org/10.1007/s11761-017-0219-8
[58] A. Verma, X. Llor, D. E. Goldberg, and R. H. Campbell, "Scaling genetic algorithms using MapReduce," in *Proc. 9th Int. Conf. Intell. Syst. Des. Appl.*, Nov 2009, pp. 13–18.
[59] S. Wadkar and M. Siddalingaiah, *Monitoring Hadoop*. Berkeley, CA: Apress, 2014, pp. 203–215. [Online]. Available: https://doi.org/10.1007/978–1-4302-4864-4_9

Carlos Guerrero received the PhD degree in computer engineering from Balearic Islands University, in 2012. He is an assistant professor of computer architecture and technology with the Computer Science Department, University of the Balearic Islands. His research interests include web performance, resource management, web engineering, and cloud computing. He has authored around 40 papers in international conferences and journals.

Isaac Lera received the PhD degree in computer engineering from Balearic Islands University, in 2012. He is an assistant professor of computer architecture and technology with the Computer Science Department, University of the Balearic Islands. His research lines are semantic web, open data, system performance, educational innovation and human mobility. He has authored in several journals and international conferences.

Belen Bermejo received the master's degree in computer engineering from Balearic Islands University, in 2015. She is a research assistant in the architecture and performance of computer and communication systems research group in the same university. She is currently doing her PhD Thesis in the field of cloud computing and resource management. She has authored some conferences papers during her master period and in the first stages of the doctoral studies.

Carlos Juiz received the PhD degree in computer engineering from Balearic Islands University, in 2001. He is an associate professor of computer architecture and technology with the Computer Science Department, University of the Balearic Islands. His research interests include performance engineering, cloud computing and IT governance. He has authored around 150 papers in different international conferences and journals. He is a senior member of the IEEE.