Multi-Objective Parallel Extremal Optimization in Processor Load Balancing for Distributed Programs

Ivanoe De Falco
Institute of High Performance Computing and Networking, CNR
Via Pietro Castellino, 111
(NA) 80131, Naples, Italy
ivanoe.defalco@icar.cnr.it

Eryk Laskowski
Institute of Computer Science, Polish Academy of Sciences
ul. Jana Kazimierza 5
01-248, Warsaw, Poland
laskowsk@ipipan.waw.pl

Richard Olejnik
Université Lille, CNRS, Centrale Lille, UMR 9189 - CRISTAL - Centre de Recherche en Informatique, Signal et Automatique de Lille
F-59000, Lille, France
richard.olejnik@univ-lille1.fr

Umberto Scafuri
Institute of High Performance Computing and Networking, CNR
umberto.scafuri@icar.cnr.it

Ernesto Tarantino
Institute of High Performance Computing and Networking, CNR
ernesto.tarantino@icar.cnr.it

Marek Tudruj
Institute of Computer Science PAS
Polish-Japanese Academy of Information Technology
Warsaw, Poland
tudruj@ipipan.waw.pl

ABSTRACT

The paper concerns multi-objective methodology applied to parallel Extremal Optimization (EO) used in processor load balancing in execution of distributed programs. When load imbalance is detected in executive processors then EO algorithms are used to find best tasks migration leading to imbalance reduction and improvement of program execution time. For this a special multi-objective version of parallel EO is applied. It is based on the EO Guided Search (EO-GS) approach which employs problem knowledge to search for the best next solution state in solution improvement. In this EO version, additional fitness function is used in stochastic selection of next solution state based on computation and communication assessment of task migration targets. In the multi-objective EO approach we jointly control three objectives relevant in processor load balancing for distributed applications. They are: computational load balance in execution of distributed applications, volume of communication between tasks on different processors and task migration parameters which fight imbalance of processor loads. The proposed algorithms are assessed by simulated execution of distributed programs macro data flow graphs.

CCS CONCEPTS

• **Mathematics of computing** → **Evolutionary algorithms**; • **Applied computing** → **Multi-criterion optimization and decision-making**; • **Computing methodologies** → *Shared memory algorithms*; *Discrete-event simulation*;

GECCO '17 Companion, July 15-19, 2017, Berlin, Germany

ACM ISBN 978-1-4503-4939-0/17/07…$15.00
https://doi.org/http://dx.doi.org/10.1145/3067695.3084218

KEYWORDS

extremal optimization, multi-objective optimization, processor load balancing

ACM Reference format:
Ivanoe De Falco, Eryk Laskowski, Richard Olejnik, Umberto Scafuri, Ernesto Tarantino, and Marek Tudruj. 2017. Multi-Objective Parallel Extremal Optimization in Processor Load Balancing for Distributed Programs. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 8 pages.
https://doi.org/http://dx.doi.org/10.1145/3067695.3084218

1 INTRODUCTION

This paper is concerned with advanced optimization methods for distributed program execution in clusters of processors by applying a nature-inspired approach called Extremal Optimization (EO) [1, 2]. EO has small computational complexity and low memory requirements which justify using this approach in processor load balancing in distributed systems. In a series of our previous papers [4–7] we have proposed and examined how EO could be applied to processor load balancing in execution of distributed programs specified as macro data flow graphs. The discussed algorithms concerned processor load balancing using both sequential and parallel single objective EO approaches. Based on these algorithms analysis and experimental results we have noticed that a multi-criteria load balancing approach could improve the algorithms by taking into account more complex optimization aims. It can be obtained by search done in a much wider solution space and by using more sophisticated methods for finding problem solutions which better match user expectations.

Good reviews and classifications of classic load balancing methods are presented in [3, 8, 9]. Surveys of load balancing methods based on evolutionary algorithms including EO are contained in [6, 7, 10]. The proposed load balancing approach is based on iterative optimization phases which improve program task placement on processors by periodic migration of tasks among processors. Parallel EO is used in iterative load balancing phases which are executed in the background in parallel with the application program.

The EO algorithms are used to discover the tasks which are candidates for migration. They are based on a special quality model used to assess task placement on executive processors. The model includes computation and communication parameters of parallel application tasks and features of the executive system components.

In these algorithms, a special parallel EO-GS approach (EO with a Guided Search) is applied in which the selection of a new partial solution to be improved is guided by some knowledge of the problem and that the parallel search of results improves the convergence rate [4]. The EO-GS approach replaces the fully random processor node selection by the stochastic selection in which the probability is more problem specific.

The algorithms presented in the current paper are improved versions of the EO-based load balancing algorithms presented in [5–7] by using a multi-objective optimization approach. Large surveys on general methods of multi-objective optimization can be found in [14, 15]. Extensive surveys on multi-objective optimization methods combined with evolutionary algorithms in general can be found in [16, 17]. These surveys are too extensive to be discussed here. Some useful papers which support the general multi-objective optimization technology are presented in [18–24]. Multi-objective approach applied to EO has already been discussed in several papers [25–29]. They propose basic methods of multi-objective optimization and cover different technical aspects of this approach. However, they are oriented towards generalized optimization problems and do not cover specific multi-objective evolutionary algorithms applied to processor load balancing.

In our previous papers on load balancing using EO approach we have considered three parameters of program execution in cluster environments: computational load of processors, inter-processor communication intensity and the number of task migrations. In the previous papers we have assumed the EO global fitness function which was a linear weighted combination of some metrics defined based on the three parameters. This approach added the burden of tuning the weight coefficients to the program and system features. In the current paper, we show that a multi-objective approach can be fruitful continuation of the research on processor load balancing presented in our previous papers. Consequently, we take a multi-objective approach and define separate objectives based on some modifications of the three mentioned above functions with unchanged general load balancing optimization axes. The three objectives have been included into a generalized EO algorithm iterative parallelized structure. The algorithm delivers the Pareto fronts of the optimization results and also the final compromise solutions obtained by finding the solutions which minimize the distance of the Pareto solutions to an ideal point with respect to a given norm. In the case of our target of load balancing, we have examined two norms: the Euclidean distance in a three-dimensional Cartesian space and the Manhattan distance.

The algorithms were assessed by experiments with simulated load balancing of distributed program represented as macro data flow graphs. The experimental results obtained by simulation compared the compromise solutions obtained using our multi-criteria parallel EO-GS approach with those of a classical EO-GS algorithm with the mentioned above singular local and global fitness functions. The superiority of the multi-objective approach against the mentioned above single-objective one has been demonstrated.

The paper is organized as follows. In Section 2 the EO principles are re-called and the proposed multi-objective parallelized EO-GS algorithm is introduced. Section 3 describes the processor load balancing approach based on the proposed multi-objective parallelized EO-GS algorithm. Section 4 presents the experimental assessment of the proposed load balancing approach applied for load balancing in simulated execution of distributed programs represented by macro data flow graphs.

2 EO ALGORITHM PRINCIPLES

2.1 EO with Guided State Changes

Extremal Optimization was proposed by Boettcher and Percus [1], following the Bak–Sneppen approach of self–organized dynamic criticality [11]. It is an attractive nature-inspired optimization method for NP–hard combinatorial and physical optimization problems.

EO operates on a single solution S consisting of a given number of components s_i, each of which is a variable of the problem and is thought to be a species of the ecosystem. Once a suitable representation is chosen, by assuming a predetermined interaction among these variables, a local fitness value ϕ_i is assigned to each of them. Then, at each time step the global fitness $\Phi(S)$ is computed and S is evolved, by randomly updating the worst variable only, to a solution S' belonging to its neighbourhood $Neigh(S)$. New solution is saved if its global fitness value is better than that of the best solution found so far.

To avoid being stuck in a local optimum, a probabilistic version of EO can be used. It is based on a parameter τ, i.e., τ–EO, introduced by Boettcher and Percus. According to it, for a minimization problem, the species are first ranked in increasing order of local fitness values, i.e., a permutation π of the variable labels i is found such that $\phi_\pi(1) \leq \phi_\pi(2) \leq \ldots \phi_\pi(G)$, where G is the number of species. The worst species s_j is of rank 1, i.e., $j = \pi(1)$, while the best one is of rank G. Then, a distribution probability over the ranks k is considered as follows: $p_k \sim k^{-\tau}$, $1 \leq k \leq G$ for a given parameter τ. At each update, a generic rank k is selected according to p_k so that the species s_i with $i = \pi(k)$ randomly changes its state and the solution moves to a neighboring one, $S' \in Neigh(S)$, unconditionally. The only parameters are the number of iterations $\mathcal{N}_{\text{iter}}$ and the selection coefficient τ.

We noticed that, when the number of neighbour states of rank k increases, the algorithm starts struggling with the problem of many possible moves. The probability of "good" state change decreases. To overcome the problem, we have proposed a modified version of EO algorithm, called Extremal Optimization with Guided State Changes (EO–GS).

In our case the term guided means that we incorporate some problem-specific information into the algorithm. τ–EO with guided state changes (EO–GS) has been proposed to improve the convergence speed of EO optimization. Some knowledge of the problem properties is used for next solution selection in consecutive EO iterations with the help of an additional local target function ω_s. It is implemented as a local target function $\omega(s)$. The value of this

Algorithm 1 EO algorithm with Guided State Changes (EO–GS)

initialize configuration S at will
$S_{\text{best}} \leftarrow S$
while total number of iterations $\mathcal{N}_{\text{iter}}$ not reached **do**
  evaluate ϕ_i for each variable s_i of the current solution S
  rank the variables s_i based on their fitness ϕ_i
  choose the rank k according to the distribution probability $k^{-\tau}$ so that the variable s_j with $j = \pi(k)$ is selected
  evaluate ω_s for each neighbour $s' \in Neigh(S)$, generated by s_j change, of the current solution S
  rank neighbours $s' \in Neigh(S)$ based on the value of target function ω_s
  choose $S' \in Neigh(S)$ according to the exponential distribution Exp(λ)
  accept $S \leftarrow S'$ unconditionally
  if $\Phi(S) < \Phi(S_{\text{best}})$ **then**
    $S_{\text{best}} \leftarrow S$
  end if
end while
return S_{best} and $\Phi(S_{\text{best}})$

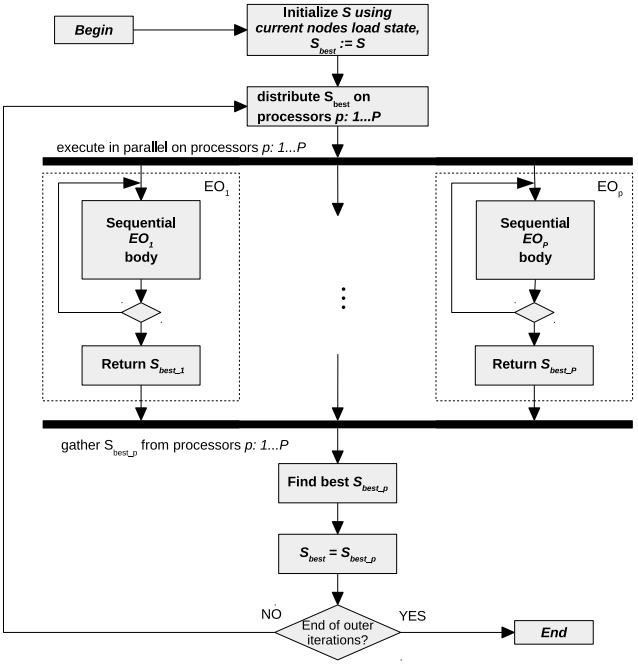

Figure 1: The general scheme of the parallel EO algorithm.

function is evaluated for all neighbours $Neigh(S)$ of rank k. Then, the neighbour solutions are sorted and assigned GS-ranks g with the use of the function ω_s. The new state $S' \in Neigh(S, s_{\pi(k)})$ is selected in a stochastic way using the exponential distribution with the selection probability $p \sim \text{Exp}(g, \lambda) = \lambda e^{-\lambda g}$. Thus, better neighbour solutions are more probable to be selected. The bias to better neighbours is controlled by the λ parameter. The general scheme of the EO–GS for minimization problems is shown as Algorithm 1.

2.2 Parallel EO general scheme

In this section we describe a parallel version of the EO algorithm that has been used in the multi-objective load balancing algorithm reported in this paper. The general scheme of this algorithm is presented in Fig. 1. The scheme begins with an initialization of the EO starting "best" solution based on current loads of all computing nodes in the distributed application. Next, a parallel part of the scheme starts, which includes iterative execution of selected versions of EO algorithms in P parallel branches.

The algorithm is constructed as a hierarchical structure of two kinds of loops: the outer main control/global data exchange loop and the inner parallel EO loops nested inside the outer loop. The inner loops are executed in parallel branches of the algorithm scheme. The inner loop body in Fig. 1 represents a sequential version of an EO algorithm executed a number of times. These can be any types of EO algorithm such as EO-GS, classic EO, or multi-objective EO which is described in the next section.

When all the parallel inner EO loops are terminated, the solution with the best value S_{best_p} among the global fitness function gathered from all parallel branches is registered as the current best one found in the current iteration of the outer loop. Next, the algorithm enters the solution exchange phase, in which an initial starting EO solution from previous iterations is identified for the next iteration of the outer loop of the algorithm. The solution is next distributed among P parallel branches of the scheme. Then, the next parallel EO algorithm outer loop iteration starts.

If the total number of EO iterations to be executed in P parallel branches is denoted as $\mathcal{N}_{\text{iter}}$ then the number of inner loop iterations in a single parallel branch is equal to $\mathcal{N}_{\text{iter}}/P$. Parallel branches are executed on separate cores of a multicore processor we use for the algorithm.

2.3 Multi-objective parallel EO-GS algorithm

In our load balancing algorithm we use a multi-objective algorithm in which we apply a number of objectives expressed as functions on the problem variable vectors which belong to a set of feasible decision vectors. A multi-objective problem can be formulated as follows: given a set of objective functions and a set of problem feasible problem variable vectors find the set of the feasible variable vectors for which the values of the objective functions are minimal or maximal. The feasible set of variables is usually defined by a set of problem derived constraints. Usually, a feasible solution that minimizes (or maximizes) all objective functions at the same time does not exist and we are satisfied by finding a so called Pareto front of solutions. A Pareto front is composed of feasible solutions which cannot be improved in any objectives without worsening of some others. Feasible solutions which belong to a Pareto front are called Pareto optimal solutions, which are based on Pareto dominance. We say that a feasible solution dominates another feasible solution if the former one is not worse than the later one in respect to all objectives but it is better than the dominated one in respect to at least one objective. The Pareto optimal solutions (i.e. belonging to the Pareto front) are not dominated by any other known feasible solutions.

Algorithm 2 Multi-objective EO algorithm with Guided State Changes (MOEO–GS)

initialize configuration S at will
$S_{\text{best}} \leftarrow S$
$D_{\text{S}} \leftarrow \emptyset$ {the set of non-dominated solutions (Pareto-front)}
while total number of iterations $\mathcal{N}_{\text{iter}}$ not reached **do**
  $C \leftarrow$ set of criteria for evaluation in the current iteration
  for all $c \in C$ **do**
    evaluate $\phi_{i,c}$ for each variable s_i of the current solution S
    rank the variables s_i based on their local fitness $\phi_{i,c}$
    choose the rank k according to $k^{-\tau}$ so that the variable s_j with $j = \pi(k)$ is selected
    evaluate ω_s for each neighbour $S_v \in Neigh(S, s_j)$, generated by s_j change of the current solution S
    rank neighbours $S_v \in Neigh(S, s_j)$ based on the target function ω_s
    choose $S' \in Neigh(S, s_j)$ according to the exponential distribution
    accept $S \leftarrow S'$ unconditionally
  end for
  if S is non-dominated **then**
    include S in D_{S}, remove dominated solutions from D_{S}
  end if
end while
select S_{best} from D_{S} using $\Phi(S)$
return S_{best} and $\Phi(S_{\text{best}})$

In our case, we want to solve processor load balancing in execution of distributed programs, given as macro data flow graphs, with the use of a multi-objective EO-GS algorithm (MOEO-GS). We have here the following problem components and data structures:

a) an EO-GS solution vector S composed of elements $s_{i,c}$, which assign program tasks to processors,
b) a set of EO local fitness functions $\phi_{i,c}$, to be used to select the solution elements for EO and MO improvements,
c) a set of EO global functions $\Phi_i(S)$ which are partial objectives of the load balancing problem and at the same time MO problem objective functions,
d) a set of non-dominated solutions NDS composed of feasible solutions which constitute the Pareto front of the MO problem,
e) a compromised MO solution which defines the set of task migrations to be done in the set of executive processors to optimally decrease processor load imbalance in the current step of the load balancing algorithm.

The pseudo-code of our generalized EO-GS-based multi-objective algorithm MOEO-GS is shown as Algorithm 2. It is iteratively executed in parallel branches of the general scheme of the parallel EO algorithms used in this paper, see Fig. 1.

The algorithm is so designed that the selection and solution improvement are performed iteratively for all objectives contained in the vector C. We operate using three objective functions oriented on supporting the load balancing problem: total computational load imbalance in execution of application tasks on processors, total volume of communication between tasks placed on different computing nodes and task migration which aims in fighting

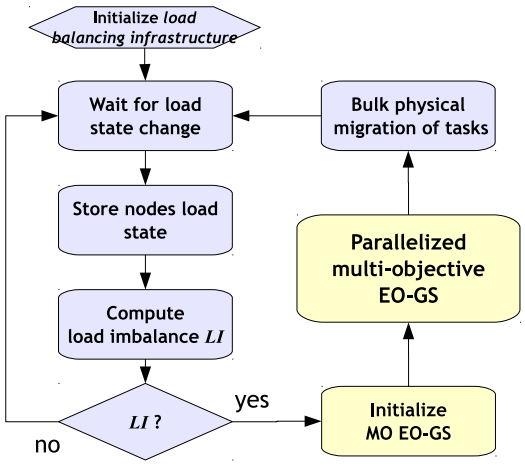

Figure 2: The general scheme of load balancing based on parallel multi-objective EO with guided state changes.

imbalance of processor loads by the possibly small number of possibly the most efficient task migrations. The way in which contents of the vector C is designed is a parameter which defines the features of the algorithm. In our design, C contains a single objective (a single EO local and a respective single global fitness function) which is selected in a probabilistic way from the three MO objectives specified for our load balancing problem mentioned above.

The Pareto front is analyzed at the end of the algorithm to deliver the S_{best} solution. The S_{best} solution will be used by the load balancing algorithm to decrease the load imbalance.

3 LOAD BALANCING BASED ON THE PARALLEL MULTI-OBJECTIVE EO-GS

3.1 Processor load balancing general scheme

The proposed load balancing method is meant for a cluster of multicore processors interconnected by a message passing network. Load balancing actions for a program are controlled at the level of indivisible tasks which are process threads.

We assume that the load balancing algorithms dynamically control assignment of program tasks $t_k, k \in 1 \ldots |T|$ to processors (computing nodes) $n, n \in 0, 1, \ldots, |N| - 1$, where T and N are the sets of all the tasks and the computing nodes, respectively. The goal is the minimal total program execution time, achieved by task migration between processors. The load balancing method is based on a series of steps in which detection and correction of processor load imbalance is done, Fig. 2. The imbalance detection relies on some run-time infrastructure which observes the state of processors in the executive computer system and the execution states of application programs. Processors (computing nodes) periodically report their current loads to the load balancing control which monitors the current system load imbalance. When load imbalance is discovered, processor load correction is launched. For this a multi-objective EO-GS algorithm (see Section 2.3) is executed to identify the tasks which need migration and the processors which will be migration targets. Next, the required physical task migrations are performed with the return to the load imbalance detection.

To evaluate the load of the system two indicators are used. The first is the computing power of a node n: $Ind_{power}(n)$, which is the sum of potential computing powers of all the active cores on the node. The second is the percentage of the CPU power available for application threads on the node n: $Time^{\%}_{\mathrm{CPU}}(n)$, periodically estimated on computing nodes. The percentage of the CPU power available for a single thread is computed as a quotient of the time during which the CPU was allocated to a probe thread against the time interval of the measurement. $Time^{\%}_{\mathrm{CPU}}(n)$ value is the sum of the percentages of CPU power available for the number of probe threads equal to the number of cores on the node.

System load imbalance LI is a boolean defined based on the difference of the CPU availability between the currently most heavily and the least heavily loaded computing nodes:

$$LI = \max_{n=0,\dots|N|-1}(Time^{\%}_{\mathrm{CPU}}(n)) - \min_{n=0,\dots|N|-1}(Time^{\%}_{\mathrm{CPU}}(n)) \geq \alpha \quad (1)$$

The load imbalance equal true requires a load correction. The value of α is set using an experimental approach (during experiments we set it between 25% and 75%).

An application is characterized by two programmer-supplied parameters, based on the volume of computations and communications tasks: $\mathrm{COM}(t_s, t_d)$ is a communication metrics between tasks t_s and t_d, $\mathrm{WP}(t)$ is a load weight metrics introduced by a task t. $\mathrm{COM}(t_s, t_d)$ and $\mathrm{WP}(t)$ metrics can provide exact values, e.g. for well-defined tasks sizes and inter-task communication in regular parallel applications, or only some predictions, e.g. when the execution time depends on the processed data.

A task mapping solution S is represented by a vector $\mu = (\mu_1, \dots, \mu_{|T|})$ of $|T|$ integers in the interval $\{0, 1, \dots |N| - 1\}$. $\mu_i = j$ means that S maps i–th task t_i onto processor j.

3.2 MOEO-GS local and global fitness functions

In this section we will describe definitions of the local and global fitness functions which are used in the proposed MOEO-GS algorithm for processor load balancing. MOEO-GS is a minimization algorithm, i.e. lower values of global fitness and lower values of local fitness are considered as better ones.

The first objective is connected to the reduction of the computational load imbalance among executive processors in the system during a given phase of distributed program execution i.e. defined by the current MOEO-GS solution S.

The global fitness functions $\Phi(S)$ for **objective 1 (computational load imbalance)** which will be minimized in the MOEO-GS algorithm:

$$\Phi_l(S) = imbalance(S) \quad (2)$$

The function $imbalance(S)$ represents the numerical load imbalance metrics in the solution S. It is equal to 1 when in S there exists at least one unloaded (empty) computing node, otherwise it is equal to the normalized average absolute load deviation of tasks in S, determined in the definition below:

$$imbalance(S) = \begin{cases} 1 & \text{exists at least one unloaded node} \\ \frac{D(S)}{D_{\mathrm{norm}}} & \text{otherwise} \end{cases} \quad (3)$$

where:

$$D_{\mathrm{norm}} = (|N| - 2) * \overline{\mathrm{WP}} + \overline{\mathrm{MINB}} \quad (4)$$

is a normalization constant, and
$D(S) = \sum_{n=0,\dots|N|-1} |\mathrm{NWP}(S, n)/Ind_{power}(n) - \overline{\mathrm{WP}}|$,
$\overline{\mathrm{WP}} = \sum_{t\in T} \mathrm{WP}(t)/\sum_{n=0,\dots|N|-1} Ind_{power}(n)$,
$\overline{\mathrm{MINB}} = \sum_{t\in T} \mathrm{WP}(t)/\min_{n=0,\dots|N|-1} Ind_{power}(n)$,
$\mathrm{NWP}(S, n) = \sum_{t\in T:\mu_t=n} \mathrm{WP}(t)$ where $\mathrm{NWP}(S, n)$ is the total computational load introduced to processor n by all program tasks allocated to n in the solution S, $\overline{\mathrm{WP}}$ is the total relative load introduced to all processors by all tasks in respect to the computing power supplied by all processors.

The local fitness function for MOEO-GS algorithm with the objective 1 is designed as follows:

$$\phi_l(t) = \gamma * load(\mu_t) + (1 - \gamma) * (1 - ldev(t)) \quad (5)$$

The function $load(n)$ indicates how much the load of node n, which executes t, exceeds the average load of all nodes. It is normalized versus the heaviest load among all the nodes. $ldev(t)$ represents the load deviation compared to the average load of all nodes. It is the absolute value of the difference between the load metrics of the task t and the minimum load on the node, normalized versus the highest such difference for all tasks on the node [6].

The second objective for the MOEO-GS algorithm is the global EO-GS fitness function $\Phi(S)$ for **objective 2 (external communication)**:

$$\Phi_c(S) = attrExtTotal(S) \quad (6)$$

The function $attrExtTotal(S)$ represents the impact of the total external communication between tasks on the quality of a given mapping S. By "external" we mean the communication between tasks placed on different nodes. This function is normalized in the range [0, 1]. In executive systems with homogeneous communication links it is a quotient of an absolute value of the total external communication volume and the total communication volume of all communications (when all tasks are placed on the same node $attrExtTotal(S) = 0$, when tasks are placed in the way that all communication is external $attrExtTotal(S) = 1$); in heterogeneous executive systems equivalent measures of the communication time are used:

$$attrExtTotal(S) = totalExt(S)/\overline{\mathrm{COM}} \quad (7)$$

where $\overline{\mathrm{COM}} = \sum_{s,d\in T} \mathrm{COM}(s, d)$ and
$totalExt(S) = \sum_{s,d\in T:\mu_s\neq\mu_d} \mathrm{COM}(s, d)$.

The local fitness function for the objective 2 is as follows:

$$\phi_c(t) = 1 - attr(t) \quad (8)$$

where the attraction of the task t to its executive computing node $attr(t)$ is defined as the amount of communication between task t and other tasks on the same node, normalized versus the maximal attraction inside the node [6].

The third objective for the MOEO-GS algorithm is concerned with task migrations induced by the current EO-GS solution S in terms of the computational load imbalance. The global EO-GS fitness function for **objective 3 (migration)** corresponds to the number of migrations:

$$\Phi_m(S) = migrationNum(S) \quad (9)$$

The function $migrationNum(S) \in \{0, 1\}$ is a migration number metrics. It is equal to 0 when there is no migration, when all tasks have

to be migrated $migrationNum(S) = 1$, otherwise $0 \leq migrationNum(S) \leq 1$ so that:

$$migrationNum(S) = |\{t \in T : \mu_t^S \neq \mu_t^{S*}\}|/|T| \quad (10)$$

where: S is the currently considered solution and S^* is the initial task placement at the start of the algorithm, μ_t^S is the current node of the task t in the solution S, and μ_t^{S*} is the node of the task t in the initial solution in the algorithm.

We have tested two variants of the MOEO-GS local fitness function for the migration objective. The first variant of the migration local objective function **LA** is based on the assessment of the quality of migrations. In this case, the definition of the local fitness function $\phi_m(S)$ is as follows:

$$\phi_m(t)^{LA} = migration_T(t) \quad (11)$$

The function $migration_T(t)$ is a quality metrics of the migration of a single task of the application. The bigger value of this function is, the worse is the migration of the task t in respect to its current node. The term "worse" migration means the migration which causes bigger load imbalance in the system:

$$migration_T(t) = |\frac{\mathrm{NWP}(S, \mu_t^S)}{Ind_{power}(\mu_t^S)} - \overline{\mathrm{WP}}| - |\frac{\mathrm{NWP}(S^*, \mu_t^{S*})}{Ind_{power}(\mu_t^{S*})} - \overline{\mathrm{WP}}| \quad (12)$$

where μ_t^S is the current node of the task t in the solution S, and μ_t^{S*} is the node of the task t in the initial solution S^* in the algorithm.

An alternative version of the local fitness function **LB** for migration objective $\phi_m(S)$ is designed to reduce the migration number regardless its influence on total processors imbalance (i.e. regardless their "quality"). Thus, the total number of migrations is reduced stronger than in the previous **LA** variant. The **LB** variant of the local fitness function for the objective 3 is as follows:

$$\phi_m(t)^{LB} = \begin{cases} 1 & \text{when task } t \text{ was migrated} \\ 0 & \text{otherwise} \end{cases} \quad (13)$$

The **LB** local fitness function forces the migration of already migrated tasks, thus increasing the probability that finally more tasks will occupy their initial computing nodes.

4 MOEO-GS ALGORITHM ASSESSMENT

4.1 Experimental setting

The experimental results have been obtained by simulated execution of application programs in a distributed system. The simulator used was built following the DEVS discrete event system approach [12]. The model of the simulated execution environment corresponds to a message-passing, distributed memory multicore parallel system, with the MPI library for communication.

During experiments we used a set of 10 synthetic exemplary programs, which were randomly generated. Their general structures resembled MPI-based parallel applications which correspond to numerical programs or physical phenomena simulations. The exemplary programs were modeled as Temporal Flow Graphs, TFG, [13]. In the TFG model, an application program consists of a set of program modules called phases, Fig 3. A phase is composed of parallel tasks which constitute process threads, and which can communicate with each other. At the boundaries between phases there is a global exchange of data possibly among all phases of the program.

The number of tasks in an application varied from 64 to 544. The communication/computation ratio C/E (the quotient of the communication time to the execution time in our experimental environment) for applications was in the range $[0.05, 0.20]$.

We distinguish regular and irregular applications. Regular applications have regular (fixed) tasks' execution times. In irregular applications the execution time of tasks depends on the processed data. Irregular applications exhibit unpredictable execution time of tasks and the communication scheme. Thus, load imbalance can occur in computing nodes even when there are no variations in computing nodes availability. In regular applications load imbalance can appear due to the non–optimized placement of tasks on processors or when runtime conditions change.

Except for task migrations, load balancing actions are performed at the background of task execution, including computations and communication. We assume also that task execution times are much longer than execution times of load balancing actions. Task migrations between processors are performed when there is no on-line communication between relevant processors involved in the migration. We assume that programs are instrumented before execution to enable task migration.

4.2 Experimental results

To study the efficiency of the presented parallel multi-objective algorithms for load balancing, we used as a reference algorithm a classic EO-GS optimization algorithm with a single objective concerning the computational load of processors (denoted in the resulting graphs as SO-C). We compared speedups and migration numbers of programs executed with load balancing based on four parallel MOEO-GS variants and also on the parallel single objective EO which used as the global fitness function a weighted sum of the three afore-mentioned optimization criteria, both in non-GS and GS version (SO-WS, SO-WS-GS). Both versions of the MOEO-GS algorithms: the "LA" version (MO-LA-GS, see equation 11) and the "LB" version (MO-LB-GS, equation 13) of local migration fitness were used in the experiments (see Section 3.2). These two versions of MOEO-GS used Manhattan distance metrics in the final utility function. Additionally, we also used the versions of MOEO-GS in which the utility function was defined using Euclidean distance metrics /MO-LA-GS(Eucl.) and MO-LB-GS(Eucl.), respectively/.

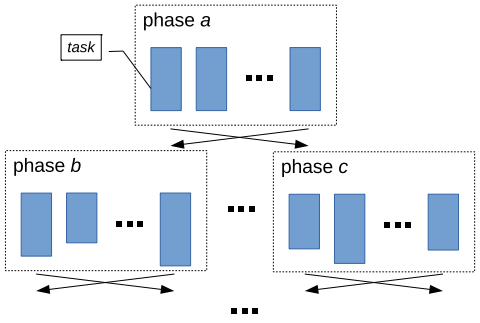

Figure 3: The general structure of exemplary applications.

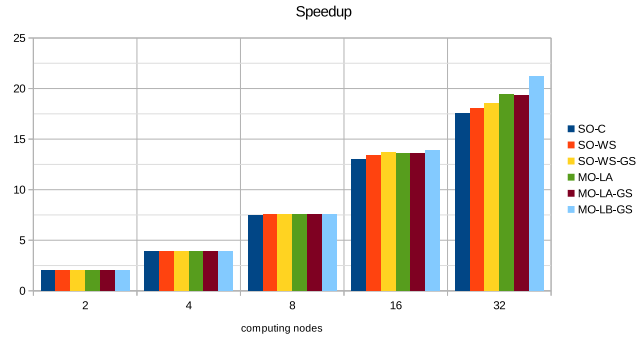

Figure 4: Speedup for different number of nodes for tested algorithms.

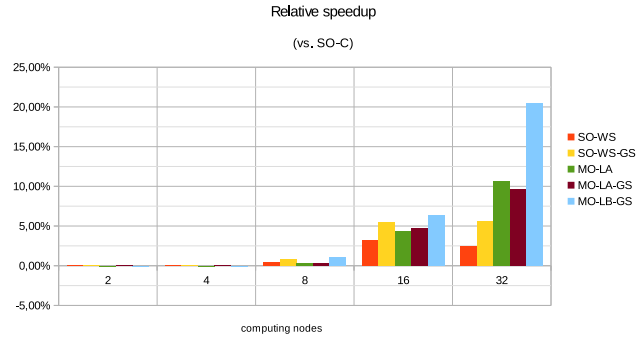

Figure 5: Relative speedup for tested algorithms versus single-objective EO.

Load-balanced execution of exemplary applications was studied in simulated systems containing from 2 to 32 homogeneous processor nodes. The following parameters for load balancing control were applied: $\alpha = 0.5, \tau = 1.5$, and $\lambda = 0.14$ for EO–GS. Additionally, the parameters $\beta = 0.5$, $\gamma = 0.75$, $\Delta_1 = 0.13$, $\Delta_2 = 0.17$ were used in the weighted sum of the global fitness function of the single objective EO-GS [4].

During the experiments described, we assumed the number of iterations for EO and MOEO $\mathcal{N}_{\text{iter}} = 500$ and the exchange rate of solutions between parallel branches every 25 inner iterations. We used $P = 4$ parallel branches, thus, the inner loop count is 25, and the outer count is 5 ($4 \times 5 \times 25 = 500$).

The results correspond to averages of 10 runs of each application. For each run 4 different methods of initial task placements (random, round-robin, METIS, packed) were tested. METIS is a multilevel graph partitioning technique which in our case creates partitions with evenly distributed computational loads and minimal cross-sections. The packed method consists in round-robin mapping of equal groups of tasks. In total, 40 runs were executed for each parameter set to produce an averaged result.

Generally, the differences between investigated algorithms increase when the number of computing nodes in the simulated system increases. The cause of this phenomenon is more demanding load balancing optimization for larger executive system (i.e. bigger possibility of the computing imbalance and harder selection of migration targets). The speedup of both parallel EO–based algorithms and the MOEO parallel algorithms as a function of the number of processors is shown in Fig. 4. The presented results are the average for irregular and regular exemplary applications. For these applications the parallel speedup obtained by multi-objective algorithms is generally greater (not worse or better) than that of a classical EO. The improvement obtained by the multi-objective approach is clearly visible in Fig. 5, which shows the relative speedup for tested algorithms versus single-objective EO. The best result over 20% is for the "LB" version of the MOEO-GS algorithm (MO-LB-GS in Fig. 5). It should be stressed that using the utility function defined as Manhattan distance (MO-LA-GS, MO-LB-GS) gives better results than Euclidean distance metrics, especially for a bigger number of processors (see Fig. 6).

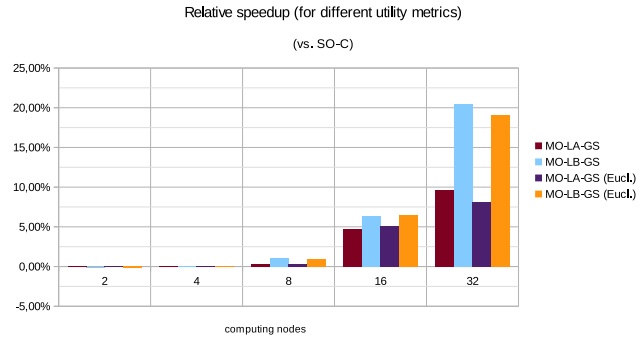

Figure 6: Relative speedup of MOEO for different utility function versus single-objective EO.

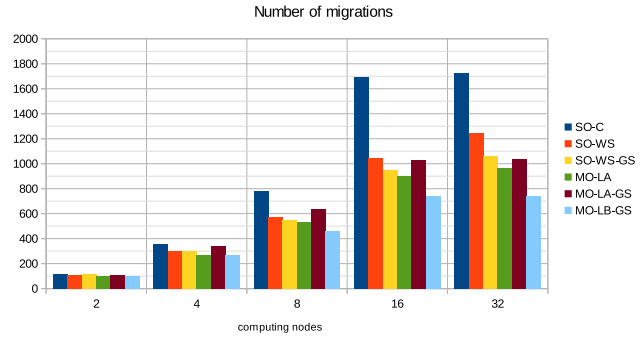

Figure 7: The number of task migrations in application execution.

Since migration costs can be very different, we decided to approximate the imposed load balancing costs by the number of task migrations, Fig. 7. The average cost imposed by multi-objective algorithms is substantially lower than the cost introduced by single-objective approaches. The reduction of the migrations number over SO-C is in the range of 25%-38% for classic EO using the weighted

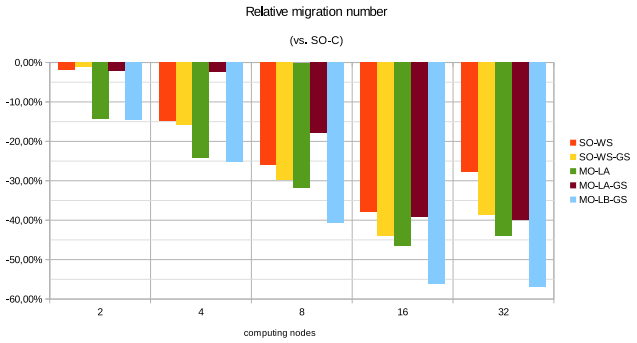

Figure 8: Relative number of task migrations versus single-objective EO.

sum in the global fitness function (SO-WS, SO-WS-GS), and in the range of 40%-56% for the multi-objective approach (MO-LA, MO-LA-GS, MO-LB-GS), Fig. 8. This confirms advantages of the proposed load balancing multi-objective algorithms.

5 CONCLUSIONS

The paper has presented a parallelized multi-objective approach based on Extremal Optimization used for processor load balancing in execution of distributed programs. Additional approach of EO-GS is embedded in the EO algorithm used in load balancing which improves its convergence. In the multi-objective EO approach, three objectives relevant in processor load balancing for distributed applications are simultaneously controlled. These objectives are: total computational load balance in execution of distributed applications, total volume of external communication between tasks placed on different processors and the number of task migrations which fix imbalance of processor loads. The task selection for EO improvements for the last objective is governed in two ways: by the decrease of weighted sum of the normalized computational loads of processors or by task migration history. The proposed algorithms were assessed by simulation experiments on EO-controlled execution of macro data flow graphs of distributed programs. The experiments have shown that the multi-objective approach added to the EO algorithms for load balancing has improved the quality of obtained results. It concerns the obtained application speedup improvement and the reduction of the migration number. A statistical algorithm assessment using hypothesis testing is foreseen in an extended version of this paper.

REFERENCES

[1] S. Boettcher, A.G. Percus, Extremal optimization: methods derived from co-evolution, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO99), San Francisco, Morgan Kaufmann, 1999, pp. 825-832.
[2] Y.Z. Lu, Y.W. Chen, M.R. Chen, P. Chen, G.Q. Zeng, Extremal Optimization: Fundamentals, Algorithms, and Applications, CRC Press, Chemical Industry Press, 2016, pp. 334.
[3] K. Barker, N. Chrisochoides, An evaluation of a framework for the dynamic load balancing of highly adaptive and irregular parallel applications, Proceedings of the ACM/IEEE Conference on Supercomputing, Phoenix, ACM Press, 2003.
[4] I. De Falco, E. Laskowski, R. Olejnik, U. Scafuri, E. Tarantino, M. Tudruj, Load Balancing in Distributed Applications Based on Extremal Optimization, LNCS Vol. 7835, EvoCOMNET 2013, Vienna, April 2013, Springer 2013, pp. 52–61.
[5] I. De Falco, E. Laskowski, R. Olejnik, U. Scafuri, E. Tarantino, M. Tudruj, Improving Extremal Optimization in Load Balancing by Local Search, LNCS Vol. 8602, EvoCOMNET 2014, Granada, April 2014, Springer 2014, pp. 51–62.
[6] I. De Falco, E. Laskowski, R. Olejnik, U. Scafuri, E. Tarantino, M. Tudruj, Extremal Optimization applied to load balancing in execution of distributed programs. Applied Soft Computing, 30 (2015), pp. 501–513.
[7] I. De Falco, E. Laskowski, R. Olejnik, U. Scafuri, E. Tarantino, M. Tudruj, Parallel Extremal Optimization in processor load balancing for distributed applications. Applied Soft Computing, 46 (2016), pp. 187–203.
[8] C. Xu, Francis C. M. Lau, Load balancing in parallel computers: Theory and Practice, Kluwer Academic Publishers, 1997.
[9] R. Z. Khan, J. Ali, Classification of task partitioning and load balancing strategies in distributed parallel computing systems, International Journal of Computer Applications 60 (17) (2012) 48–53.
[10] M. Mishra, S. Agarwal, P. Mishra, S. Singh, Comparative analysis of various evolutionary techniques of load balancing: a review, International Journal of Computer Applications 63 (15) (2013), pp. 8–13.
[11] K. Sneppen, et al.: Evolution as a self–organized critical phenomenon. Proc. Natl. Acad. Sci. 92 (1995), pp. 5209–5213.
[12] B. Zeigler, Hierarchical, modular discrete–event modelling in an object-oriented environment, Simulation 49 (5) (1987), pp. 219–230.
[13] C. Roig, A. Ripoll, F. Guirado, A new task graph model for mapping message passing applications, IEEE Trans. on Parallel and Distributed Systems 18 (12) (2007), pp. 1740–1753.
[14] Y. Collette, P. Siarry, Multi-objective Optimization: Principles and Case Studies Springer, 2004, pp. 293.
[15] M. Ehrgott, Multi-criteria Optimization, Springer, 2005, pp. 324.
[16] C. A. Coello Coello, Evolutionary Multi-Objective Optimization: A Historical View of the Field, IEEE Comp. Intelligence Magazine, Feb. 2006, pp. 28-36.
[17] C. A. Coello Coello„ G. B. Lamont, D. A. Van Veldhuizen, Evolutionary Algorithms for Solving Multi-Objective Problems, Springer, 2007, pp. 800.
[18] J. D. Knowles, D. W. Corne, The Pareto archived evolution strategy: A new baseline algorithm for multi-objective optimization. The 1999 Congress on Evolutionary Computation, Piscataway, NJ, IEEE Press 1999, pp. 98-105.
[19] J. D. Knowles, D. W. Corne, Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy, Evolutionary Computation 8(2), MIT Press, 2000, pp. 149-172.
[20] P. L. Yu. A class of solutions for group decision problems. Management Science, 19(8):pp. 936-946, 1973.
[21] C. Busing, K-S. Goetzmann, J. Matuschke, Compromise Solutions in Multicriteria Combinatorial Optimization, Technical Report TU Berlin, 9-2011, pp. 27.
[22] K. Deb, M. Mohan, and S. Mishra, Evaluating the E-Domination Based Multi-Objective Evolutionary Algorithm for a Quick Computation of Pareto-Optimal Solutions, Evolutionary Computation, Vol. 13, N. 4, 2005, pp. 501-525.
[23] K. Deb, M. Mohan, S. Mishra, Towards a quick computation of well-spread pareto-optimal solutions. Second Evolutionary Multi-Criterion Optimization Conference, EMO-03, LNCS Vol. 2632, 2003, pp. 222-236.
[24] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler, Combining convergence and diversity in evolutionary multi-objective optimization, Evolutionary Computation, 10(3), 2002, pp. 263-282.
[25] E. Ahmed, M. F. Elettreby, On multi-objective evolution model. International Journal of Modern Physics C 2004; 15 (9); 1189-1195
[26] P. Gómez-Meneses, M. Randall, A. Lewis, A Hybrid Multi-objective Extremal Optimisation Approach for Multi-objective Combinatorial Optimisation Problems, Bond University, Griffith University, Australia, 2010.
[27] R. L. Galski, F.L. de Sousa, F. M. Ramos, I. Muraoka, Spacecraft thermal design with the generalized extremal optimization algorithm. Proceedings of Inverse Problems, Design and Optimization Symposium, Rio de Janeiro, Brazil, 2004.
[28] M. Chen, Y. Lu, G. Yang, Multi-objective extremal optimization with applications to engineering design, Journal of Zhejiang University SCIENCE A, Vol. 8 (12) (2007), pp. 1905-1911.
[29] I. De Falco, A. Della Cioppa, D. Maisto, U. Scafuri, E. Tarantino, A multi-objective extremal optimization algorithm for efficient mapping in grids, Applications of Soft Computing, Advances in Intelligent and Soft Computing 58, 2009, Springer, pp. 367-377.