Double-layer evolutionary algorithm for distributed optimization of particle detection on the Grid

To cite this article: Adam Padée et al 2013 J. Phys.: Conf. Ser. 454 012084

View the article online for updates and enhancements.

Related content

- Particle detection optimization by means of a distributed evolutionary algorithm
  Adam Padée, Krzysztof Zaremba and Krzysztof Kurek

- Operating the Worldwide LHC Computing Grid: current and future challenges
  J Flix Molina, A Forti, M Girone et al.

- Status of the DIRAC Project
  A Casajus, K Ciba, V Fernandez et al.
Double-layer evolutionary algorithm for distributed optimization of particle detection on the Grid

Adam Padée 1,2, Krzysztof Kurek 3 and Krzysztof Zaremba 1

1 Warsaw University of Technology, Institute of Radioelectronics, ul. Nowowiejska 15/19, 00-665 Warsaw, Poland
2 National Centre for Nuclear Research, Świerk Computing Centre, ul. Andrzeja Soltana 7, 05-400 Otwock-Świerk, Poland and
University of Warsaw, Interdisciplinary Centre for Mathematical and Computational Modeling (ICM), ul. Pawińskiego 5a, 02-106 Warsaw, Poland
3 National Centre for Nuclear Research, Theoretical Physics Department, ul. Andrzeja Soltana 7, 05-400 Otwock-Świerk, Poland and
Cardinal Stefan Wyszyński University in Warsaw, Faculty of Mathematics and Natural Sciences, The School of Exact Sciences, ul. Wóycickiego 1/3, 01-938 Warsaw, Poland

E-mail: adam.padee@ncbj.gov.pl

Abstract. Reconstruction of particle tracks from information collected by position-sensitive detectors is an important procedure in HEP experiments. It is usually controlled by a set of numerical parameters which have to be manually optimized. This paper proposes an automatic approach to this task by utilizing evolutionary algorithm (EA) operating on both real-valued and binary representations. Because of computational complexity of the task a special distributed architecture of the algorithm is proposed, designed to be run in grid environment. It is two-level hierarchical hybrid utilizing asynchronous master-slave EA on the level of clusters and island model EA on the level of the grid. The technical aspects of usage of production grid infrastructure are covered, including communication protocols on both levels. The paper deals also with the problem of heterogeneity of the resources, presenting efficiency tests on a benchmark function. These tests confirm that even relatively small islands (clusters) can be beneficial to the optimization process when connected to the larger ones. Finally a real-life usage example is presented, which is an optimization of track reconstruction in Large Angle Spectrometer of NA-58 COMPASS experiment held at CERN, using a sample of Monte Carlo simulated data. The overall reconstruction efficiency gain, achieved by the proposed method, is more than 4%, compared to the manually optimized parameters.
1. Introduction
Detection of momenta of charged particles plays an important role in High Energy Physics (HEP) experiments. It is also one of the most complex steps of the data processing chain, because it requires reconstruction of the tracks of particles in 3 dimensions, on the basis of signals from a set of position sensitive detectors. Physical limitations of the detectors (precision, efficiency) make it necessary to control the process with parameterized tolerances. Setting of these parameters can be treated as an optimization task. Although there is no precise formula to evaluate the quality of each parameter set, it can be estimated using Monte Carlo simulation of the experiment. The drawback of using this kind of criterion is the lack of precision. The quality of the reconstruction is expressed in the ratio of properly reconstructed tracks to all tracks existing in the generated sample, so it may not reflect the effects of small changes in the parameters. The criterion is also dependent on the exact events that were generated, so it is only statistically correct when using different samples. These features of the objective function make the problem hard to solve using standard numerical optimization methods.

Evolutionary algorithms have been extensively used in many scientific and engineering applications over last decades. Their unique features make it possible to solve problems that are difficult for classical optimization methods. One of the main advantages of evolutionary algorithms is that they do not rely directly on any derivative-based information (e.g. gradient) [1]. Therefore they can be applied to problems where objective function is either imprecise or distorted by a random noise. The application described in this article belongs to this class of problems, because of the random nature of the evaluation process. This is the reason why evolutionary algorithms have been chosen as the optimization method.

One of the major difficulties of using evolutionary optimization is the relatively high demand for the computing power, at least in comparison to the analytical methods. This is the price for robustness coming from ignoring the derivative-based information [1]. The problem is even more severe when the objective function itself is computationally expensive, as it is with the reconstruction of Monte Carlo data, which is the application described in this paper. On the other hand, evolutionary algorithms are relatively easy to parallelize, because at each step they operate on large populations instead of individual points. This feature makes them well suited for usage in a broad spectrum of parallel and distributed environments. There were many industrial and scientific applications of evolutionary algorithms that make use of supercomputers, clusters or even computational grids. One of such applications is the previous version of the program presented in this paper, which solved the problem of reconstruction optimization using master-slave parallel genetic algorithm designed for cluster of PCs. Detailed description of that program can be found in [2]. The results achieved by that version were not entirely satisfactory, mainly because of lack of computing power. This was the inspiration for the creation of the algorithm described in this paper, designed to use computational grid instead of single cluster.

2. Objective function
The application described in this article was created to support the muon program of NA-58 COMPASS (Common Muon-Proton Apparatus for Spectroscopy and Structure) experiment that is currently held at CERN. Further details on the experiment can be found in [3] and [4]. As the work described in this article is a continuation and extension of the application described in [2], only a short summary of the information on momentum measurement, data reconstruction process and the objective function will be presented.

The measurement of the momentum of high-energy charged particles is done by checking the curvature of their tracks in a strong magnetic field. There are two sets of position-sensitive detector planes and a magnet between them. The data from the detectors is registered in raw format and, due to computational complexity, further processing is done offline. The reconstruction of the tracks is a major part of this processing and it can be split into the following steps:

- Conversion of the raw signals to „hits” (2D positions on the detector planes plus timestamps).
Reconstruction of straight segments of the tracks in the areas with the detectors (the procedure assumes that the magnetic field in these regions is zero).

- Linking of the segments into tracks, extrapolating them in the magnet area.
- Finding the interaction vertex (common point from which the tracks originate)

There are several limitations that decrease the efficiency and precision of the procedure. The most notable examples are: the efficiency of the detectors, limited spatial and temporal resolution of the planes, high number of tracks in one event, background noise, pile-up tracks and simplified mathematical models (e.g. assumption that the segments in the detector areas are straight, which is not true because of fringe magnetic field in that area). This is why the reconstruction process is done iteratively, and each of the iterations is controlled by separate set of parameters, e.g. tolerances for the straight parts of the track (in both detector sets), the precision of the crossing point in the magnet area, tolerance for the interaction vertex and minimum number of hits needed to qualify a track as good. Although their values can be narrowed down to certain ranges by analyzing the physical properties of the detector, the choice of the exact values have to be done experimentally using Monte Carlo simulation of the detector. In COMPASS the detector is simulated using Geant with input from the event generators (PYTHIA, AROMA, LEPTO). The offline processing program is called CORAL (COmpass Reconstruction and anALysis).

The quality of particular set of parameters is expressed with the following criteria: the number of properly reconstructed tracks (denoted by $\Phi_1$) and the number of false tracks introduced by the reconstruction routine (denoted by $\Phi_2$). The overall criteria used in the algorithm is:

$$\Phi = \Phi_1 - c \cdot \Phi_2$$  \hspace{1cm} (1)

where $c$ is an arbitrarily chosen constant (in the experiments it was set to 1.5). Additionally the reconstruction time is checked, if it exceeds a certain threshold, CORAL is aborted and the objective function is zeroed. More details on the reconstruction can be found in [5], [4]. The objective function is described in more detail in [2].

3. Computational grid in HEP

The growing complexity of HEP experiments results in higher demands for computing power. They often exceed the capabilities of the computing farms dedicated to the experiments. This is the reason why HEP is shifting towards distributed data analysis using computational grids. The best examples of this approach are the experiments built around LHC (Large Hadron Collider), which established their own grid environment named WLCG (Worldwide LHC Computing Grid [6]). Although the project itself does not provide any resources, it collaborates with many national and continental grid initiatives to make their infrastructures available for physicists and ensure the availability of a standard set of middleware components and protocols. Standards set by WLCG are widely accepted also by smaller, non-LHC experiments, COMPASS being one of them. The access to the resources in the grid uses the concept of Virtual Organizations (VOs). Each VO corresponds to the community of researchers. In HEP a separate VO is usually set up for each experiment. It is up to the resource providers to decide which VO to support and how big share of their resources they are granted. The authorization of the individual users and the verification of their VO membership are done using x509 certificates.

There are two main types of the resources accessible via the grid: CPUs and storage. Computing resources in WLCG are served by machines called Computing Elements (CEs). Each CE is a frontend service to a cluster of Worker Nodes (WNs) managed by a queuing system (Torque, LSF, SGE, ...). The user does not need to know what type of a queuing system lies underneath a specific CE, because jobs are submitted using standard API or CLI. Storage resources are served by machines called Storage Elements (SEs). There are couple of types of SEs, but despite the differences in hardware and storage capacity, all of them offer similar set of protocols. The management of the storage space is done via SRM (Storage Resource Manager) protocol. File uploads and downloads are performed using
GridFTP standard. Each of the SEs also offers its own implementation of a low-level random I/O protocol (e.g. RFIO, dcap). Although they are different, there is a common API called GFAL (Grid File Access Library) that offers unified access to data with POSIX-like interface, hiding the actual access method from the user. Real life tests with WLCG infrastructure show that the delay for a single transaction with small file varies from 1.5 s to couple of seconds for GridFTP (depending on the distance between the communicating parties), and from 0.5 s to couple of seconds when using GFAL.

The information about the resources connected to the grid is gathered in a common information system called BDII (Berkeley Database Information Index). It contains both static (like a number of CPUs in a CE, total storage space of a SE, supported VOs) and dynamic data (e.g. current usage). Using this system the resources available to a VO can be easily discovered and centrally managed, which is done by the collection layer services. The optimal distribution of the load between CEs is managed by WMS (Workload Management Service). Unified access to all the storage resources available to a VO is possible using LFC (Logical File Catalogue), which constitutes a global virtual namespace, resembling a single filesystem. The names of files in LFC, called LFNs (Logical File Names) can be used directly in GFAL API calls.

Typical usage pattern of the grid by HEP communities is the submission of series of sequential jobs which share data sets using SEs. In these models there is no need to provide any direct communication protocol between the jobs, especially when they are running on different clusters. For this reason WLCG does not provide any tools for running inter-cluster parallel jobs. There are some software components compatible with WLCG middleware, developed by external projects (e.g. Int.EU.Grid [7]), that enable parallel communication between processes running at different clusters, but they did not become a widely accepted standard yet. Even the support for MPI jobs limited to a single cluster is not compulsory for the WLCG member sites and remains an optional feature.

The facts outlined in the paragraph above are a huge barrier for design and usage of the parallel algorithms that could use all the resources available in the grid in a coordinated way. Still this is potentially a very beneficial approach, because the large fraction of the computational resources in the grid is provided by many small clusters. If the numbers of CPU cores provided by the smallest clusters are summed up, they often prove to be larger than the contribution of the largest clusters. The example of such a situation can be seen in table 1, which contains distribution of the cluster sizes in EGI (European Grid Infrastructure), which is one of the major contributors of resources to WLCG.

| Size range     | 0-1024 | 1025-2048 | 2049-4096 | 4097-8192 | 8193-16384 | 16385+ | Sum  |
|----------------|--------|-----------|----------|----------|-----------|--------|------|
| Total cores    | 33346  | 32870     | 70615    | 57661    | 32476     | 16484  | 243452 |
| No of clusters | 167    | 22        | 24       | 9        | 3         | 1      | 226   |

None of the categories presented in the table is predominant. Although differences between the categories are significant, their contributions have the same order of magnitude. Designing an algorithm capable of using only large clusters would waste a considerable part of the resources. It was therefore essential to find an architecture capable of using the resources distributed in the smallest clusters. An additional requirement was to use file transfer protocols, as they are the only standard option for inter-cluster communication.

4. Evolutionary algorithms in numerical optimization

Evolutionary Algorithm (EA) is a general name for a class of metaheuristic optimization methods. Their common trait is that they use mechanisms inspired by biological evolution: selection, reproduction, mutation and recombination (crossover). Detailed description of EAs can be found in [8], so here only a short summary of the most important facts will be given. EAs are iterative algorithms, but unlike the classical optimization methods, they operate on $N$-element population in each step, instead of a single point. This is the main reason why they are relatively easy to parallelize,
because many time-consuming operations can be done independently for each of the population members. An iteration is often called a generation. EAs use specific representations of the points in the search space, which are often called genotype or chromosome. The choice of representation is the most distinctive feature of different EA types, because it also determines the way in which the genotype is modified by mutation and crossover operators. In EAs used for numerical optimization, there are two main types of representations, historically originating from different types of algorithms:

- Genetic algorithm. Optimized parameters are coded as binary strings. Each parameter is sampled evenly within allowed range with desired precision and represented as a fixed-point number (sometimes using Gray code). Mutation and crossover operate on individual bits.
- Evolution strategy, evolutionary programming. Optimized parameters are coded as real values, usually with floating-point representation. Mutation adds a small perturbation with Gaussian distribution to the processed parameter. Crossover can exchange only whole parameters; in some implementations (called mixing crossover) it can also calculate weighted average of the respective parameters.

In the reconstruction optimization both representations are useful, because binary coding is better for enumerative parameters, and real better suits continuous variables. As stated in [1] there is a rationale that binary coding of variables for which the objective function has periodic component may be also beneficial. Therefore both representations were implemented in the program. More details on how the reconstruction parameters are coded in the genotype can be found in the description of the previous version in [2], because this part of the program did not change significantly. More general discussion about the representations and operators can be found in [8].

The most general rule of EAs is that better points should have more copies in subsequent generations than worse ones. This is ensured by the selection and reproduction operators. They usually do not work directly on the objective function values, but use its modified form, called fitness function. The conversion from the original objective function to the fitness function involves simple operations like negation (necessary for minimization problems) and/or linear scaling to ensure that the values are in the proper range (e.g. to avoid negative values which are not acceptable by proportional selection). The optimization of the reconstruction is a maximization problem, so it does not require modifications except simple linear scaling. The genotype together with a corresponding fitness value is called an individual in EA terminology.

5. Architectures of parallel evolutionary algorithms

According to [9] and [10] there are three main types of parallel evolutionary algorithms: master-slave, cellular and multiple-population. The simplest and the most popular one is master-slave. In this variant all the evolutionary operators are executed centrally within the single master process. Only the fitness calculation is distributed and delegated to multiple slave processes. The popularity of this variant is based on the fact that calculation of the fitness values is usually the most time-consuming part of the EA iteration. If the algorithm is executed in \( k \)-processor environment, \( T_g \) denotes time needed to execute the evolutionary operators for a single individual, \( T_c \) time of communication with one slave process and \( T_f \) time of fitness calculation for single individual, the overall time needed to perform a single iteration of the algorithm \( T_p \) is given by:

\[
T_p = NT_g + kT_c + \frac{NT_f}{k}
\]  

In optimization of the reconstruction the assumption that \( T_f \gg T_g \) is fulfilled, but using grid protocols for communication results in very high \( T_c \). Theoretical scaling boundary of the master-slave algorithm can be calculated as:

\[
k^* = \left( \frac{NT_f}{T_c} \right)^{\frac{1}{2}}
\]
Increasing the number of processors above $k^*$ will increase the total execution time of an iteration ($T_p$) instead of decreasing it. Moreover, it has to be noted that the algorithm operating with $k=k^*$ is very inefficient, and almost half of the CPU time is wasted on waiting. E. Cantú-Paz in [10] calculated (with some simplifying assumptions) the efficient scaling boundary $k_i$, where the wasted CPU time equals roughly $1/k$ (the equivalent of 1 CPU core is wasted).

\[
k_i = \left(\frac{NT_p}{T_c}\right)^{\frac{1}{k}}
\]

(4)

Analyzing a real case of optimization of the reconstruction in the grid environment, $T_f$ is about 3-10 minutes (depending on the sample size and the particular values of the parameters) and $T_c$ 1-3 seconds. With 200-element population, $k_i \approx 30$ and $k^* \approx 200$ which makes the practical usability of the algorithm very limited.

There is also another problem concerning the classical master-slave design, which is variable $T_f$. In the analyzed case it depends not only on the particular parameter settings, but also on the differences in CPU processing speeds in different grid clusters. The master process have to wait for the slowest slave, so the expected duration of the fitness calculation phase for the whole algorithm will be much higher than the mean value of $T_f$ multiplied by $N/k$ (for simplification it is assumed that $N$ is divisible by $k$ without the remainder and the time spent by a single slave on the fitness calculation is denoted by $T_{fs}$). If $f(T_{fs})$ denotes the probability density function of $T_{fs}$ and $F(T_{fs})$ cumulative density function, then the probability density of the duration of fitness calculation phase on $k$ processors will be given by the first member of the order statistics:

\[
f_{MAX}(T_{fs}, k) = k \cdot f(T_{fs}) \cdot [F(T_{fs})]^{k-1}
\]

(5)

The exponential relation between the overall generation time PDF and slave calculation time CDF confirms the intuitive conviction that $f_{MAX}(T_{fs}, k)$ will rise very quickly when increasing the number of processors.

To overcome both problems outlined in the paragraphs above a special asynchronous version of master-slave algorithm can be used. It doesn’t have clearly marked generations when whole parent population is replaced by offspring population, but instead new individuals are constantly generated, sent to slaves and immediately inserted to the current population whenever they are returned. This requires the introduction of a new operator – succession. When new individual is received from a slave, this operator decides which place in the population it will be inserted to. This choice can be random or controlled by multiple criteria e.g. fitness measures, similarity of the genotypes, lifetime, etc.. As a matter of fact this type of the algorithm, called steady-state EA (abbreviated to ssEA), does not have to be connected with parallelism. It was invented and used first in a sequential version [14].

Asynchronous master-slave EA is not as efficient as previously described synchronous variant when it is used with low $k$. The reason is that in this model the master process does not do any fitness calculations itself, because it has to be constantly ready for the new individuals coming from the slaves. Therefore the resource usage efficiency has always an upper bound of $(k-1)/k$, regardless of the $T_f$, $T_g$ and $T_c$. Nevertheless, with higher $k$ one can expect higher efficiency, because the time $T_{pa}$ needed by the algorithm to process $N$ individuals (which is equivalent to processing one generation in the synchronous variant) is expressed by:

\[
T_{pa} = T_c + \frac{NT_f}{k-1}
\]

(6)

Although this equation sets the practical scaling boundary much higher than the value $k_i$ for the synchronous variant, the absolute boundary of $k^*$ stays the same, because the equation (6) holds up only within the range given by the following inequality:

\[
T_g + \frac{(k-1)T_f}{N} < \frac{T_f}{k-1}
\]
Still the asynchronous variant is much better for higher number of CPUs, because it is practically usable up to $k^*$ and its efficiency of resource usage at that threshold is close to $(k-1)\cdot N/k \cdot (N+1)$, which is much higher than $\frac{1}{2}$. It is also not sensitive to random variations of $T_f$. For these reasons it has been used in the previous version of the optimization program, designed to run on a single cluster, which is described in [2]. Nevertheless, the scaling boundary of $k^*$ is too low to use it efficiently in the grid. Higher number of the processing units can be used only in other variants of parallel EAs.

Cellular algorithm is not suitable for the grid, because it is communication-intensive and requires homogenous execution environment. Better results could be achieved with multiple-population algorithm, which is also called island model EA. This is a fully distributed algorithm, in which each processing unit holds its own subpopulation. The number of subpopulations (also called islands or demes) is denoted by $r$. The algorithm works in two phases. In the first one all the subpopulations evolve independently, so it is equal to $r$ independent sequential algorithms. In the second phase the demes exchange selected individuals. This is done by an additional operator named migration. The connections between the islands form a topology graph. Topologies are mostly regular graphs, described by $\delta$ parameter, which denotes the degree of each vertex. It is useful when comparing different topologies, as it is closely related with the time spent on communication. Other parameters that control the island model EA are: migration coefficient $\rho$, which defines percentage of individuals in each population that are exchanged in the migration phase, interval between the migration phases (also called epoch – measured in number of generations and denoted by $g$), number of individuals in each subpopulation ($N_d$) and number of epochs ($r$). As it has been shown in [10], $\rho$ and $\delta$ do not have to be high to achieve the convergence speed comparable to single population with $r \cdot N_d$ individuals.

There is no absolute scaling boundary for island model EA, because communication to local calculations ratio is set by parameters $g$, $\rho$ and $\delta$, which are defined arbitrarily by the user. Still there are some ranges in which these parameters should be contained in order to achieve good convergence speed. For example if the subpopulations are small, it makes little sense to set the epoch length too high, because at some point each subpopulation will be taken over by one dominant solution and from that moment further local calculations are pointless. In distributed EAs there is a general rule that the communications frequency should be inversely proportional to the size of subpopulations. This relation is often depicted in the form of 3-dimensional plot, binding the communication frequency, the amount of local calculations and the number of subpopulations [10], [9]. The plot is presented on figure 1. Optimally configured algorithms are placed on space diagonal of the cube. On one end of the diagonal there is sequential algorithm, on the other – cellular. All the configurations in between are island model EAs with different settings of the parameters. Using the island model EA with grid resources would be difficult, because a big number of small islands require frequent communication. Possible solution to this problem would be to use hybrid algorithm which is composed of two layers of communication and uses different architecture on each of them.

![Figure 1. Relation between communication frequency, local calculations and number of subpopulations in the distributed EA.](image-url)
6. Hybrid EA for the grid

The idea behind the design of hybrid, double-layer parallel EA is to use multiple parallel EAs and connect them in a bigger structure using the migration operator. EAs on the lower level could be of any type (master-slave, cellular or multiple-population) and possibly they could use different communication protocol than used by the migration on the higher level. The last feature is important when running the algorithm on non-uniform resources and allows adapting it to their physical layout (fast interconnect is used within the islands, slow one between them). The main advantage of the hybrid algorithm over the flat island model is that thanks to parallelism on the level of the islands it is possible to make them much bigger than in sequential case. This allows to set a lower migration intensity so the usage of slow interconnect can be lowered by orders of magnitude. Hybrid EAs have been successfully used in many applications over last two decades, there are also good performance analyses of this type of algorithm, e.g. in [12]. The facts outlined above make this algorithm perfectly suited for usage in the grid environment. Islands can be formed on whole clusters instead of single CPU cores and use fast MPI interconnect for internal communication wherever possible. Still there are several major problems when adapting hybrid model EA for the grid. First, the clusters have different sizes, varying from tens to thousands of cores. Although large clusters usually support more VOs, so the partition available for each of them is relatively smaller, but still the algorithm should be able to work with differences in the processing speeds of the islands reaching one or two orders of magnitude. Second problem is that the availability of the communication protocols is different over the grid. Not all of the clusters enable MPI usage and not all of the SEs allow the external clients (outside their own subnet) to use random I/O protocols (RFIO, dcap). The algorithm should allow using different protocols on both levels and be resistant to unpredictable communication delays.

The problems described above implicate the usage of asynchronous communication model both locally and globally. It is relatively easy to implement when using master-slave ssEA at the islands. The master process in ssEA is always ready to accept a newly rated individual from any of the slaves. Migrating individuals from other subpopulations can be treated in the same way. Using this design there is no need to synchronize migrations and local calculations phases between the islands. Migrants can be incorporated to any of the subpopulations at any time, and the local population benefits from their presence immediately (which is better than using the migration buffers).

The problem of huge differences in islands’ processing speeds is a more serious one, and it can have potentially negative influence on the convergence of the whole algorithm. There are no theoretical approaches to this problem so far in the literature concerning parallel evolutionary algorithms. However, there is a thorough experimental study by J. Branke et al. in [13], where the authors analyze islands with processing speed differences more than one order of magnitude (the fastest one is 32 times faster than the slowest one). The tests are done with several topologies and migration strategies. The results show that such big differences in processing power of the islands indeed have negative impact on the convergence speed when compared to the uniform environment of the same summary size. The conclusion from the work is that the negative influence of non-uniform distribution of the resources can be minimized by using a topology where big and small islands are mixed (size differences between neighbours are maximized). Better results were also achieved when the migration was initiated by the receiving island (import strategy).

Another problem, specific for the grid environment, is the instability of the resources. There is no central resource manager (WMSes are not synchronized), so transient changes in the load of the clusters (including total congestion) are impossible to predict. Whole clusters can also appear and disappear from the information system (e.g. due to maintenance or failures) so the algorithm should be able to adapt to the actual state of the grid. It is therefore not advisable to use a static topology, especially with low \(\delta\) (which is desired taking into account the slow interconnect), because the graph could even disintegrate. This was addressed in the algorithm by selecting random edges from fully connected topology at each migration. Dynamic changes of the computing power of the clusters have also another side effect. They should be reflected in appropriate changes of the migration intensity to maintain the optimal proportion between the communication and local calculations, like it is depicted
on figure 1. Moreover, even without these changes, it would be a big burden on the user of the algorithm to tune the migration rates for each island manually, depending on the size of the island, number of islands and the optimized function. To automate this process it is possible to use adaptive migration. The idea is to run the migration operator when some criterion of convergence is met. The most popular indicator used in this kind of algorithms is based on changes of standard deviation of the fitness in the population. It has been proposed by Munetomo et al. in [11]. The standard deviation in the first generation is calculated and saved as $\sigma_0$. In every subsequent generation the current value of $\sigma$ is calculated and the algorithm checks if $\sigma/\sigma_0$ did not fall below certain threshold (chosen arbitrarily between 0.1 and 0.3, depending on the optimized problem). If it is lower, the migration is performed. The algorithm can be also used in ssEA, despite that there are no designated generations in it – $\sigma_0$ is calculated immediately after first $N_d$ individuals are collected, and subsequent checks of $\sigma/\sigma_0$ are done in regular intervals after each $N_d$ processed individuals.

7. Technical details of the implementation

The algorithm outlined above has been implemented as an extension of the previous version, presented in [2]. The interface with CORAL works in the same way: a temporary file with the parameters is created and CORAL is run as an external program. The algorithm keeps track of all the running CORAL instances in the internal structure called execution queue. Fitness is calculated by parsing text output from CORAL and assigned to the proper genotype basing on the index in the queue. The master-slave part though uses different scheme of communication than its predecessor. The previous mechanism of dynamically launched slaves via cluster queuing system for each individual proven to be inefficient on heavily loaded clusters without special high-priority queues for short jobs. This was the case for the majority of the resources, because most of the clusters have just one queue per VO. Therefore the mechanism was changed to persistent slaves. Their communication with the master no longer uses queuing system mechanisms, but was changed to one of the following options:

- **MPI.** This is the preferred interconnect. The master process uses synchronous sends and asynchronous receives. The program can use any flavour of MPI (tested with OpenMPI 1.x and MVAPICH v2). To avoid version mismatches, the program can be compiled with local MPI library after submission to the target cluster in pre-run hook. No scaling problems were observed up to 1500 slaves, which was the maximum cluster share available (conditions of the test: DDR Infiniband network as a local interconnect, test fitness function with $T_f = 1s$).
- **GFAL.** Backup interconnect option for clusters not having MPI. Local SE has to be given in the program options (automatic discovery was not implemented yet). Practical scaling boundary measured for DPM RIIO is about 150-170 slaves for the production fitness function, close to $k^*$ estimated in chapter 5.

The upper layer of the algorithm works in fully connected logical topology with random drawing of the edge, which will be used in a particular migration call. In case of failure the procedure is repeated and another edge is drawn. Mutual discovery of the islands is addressed by using a central directory either on specified SE (faster, vulnerable to SE failure) or in LFC filesystem (slower, but with the possibility of creating replicas on different SEs). Total number of the islands has to be defined statically in the options before running the algorithm, but not all of them have to start at the same time. Migrations can use either of the two interconnects:

- **GFAL.** Requires the SE (or SEs) selected for migration to allow global incoming connections to random I/O ports (RIIO, gsidcap), so in certain configurations it may not work.
- **GridFTP.** Uses GridFTP protocol, slightly slower than random I/O but accessible for grid-wide connections on all the SEs. Actual transfers are performed by external client (uberftp), which is part of standard WN installation.
8. Test on a benchmark function

Initial tests were performed on slightly modified 10-dimensional Rastrigin function. It is popular benchmark to test optimization methods. The function is composed of low frequency paraboloid component modulated by higher frequency periodic function, so it has many local optima. The exact formula that was used in the test is following:

\[
\Phi_{test}(x_1,\ldots,x_{10}) = 100 - \sum_{i=1}^{10} [x_i^2 - 10 \cdot \cos(2\pi x_i)]
\]  

(7)

All the variables were limited to \(-10,10\). The main goal of the test (apart from verifying the performance of the algorithm on a well-known multi-modal problem) was to check how the slow interconnect and the non-uniform distribution of the resources affect the optimization process. It was not meant to examine all the possible configurations and topologies, because this kind of tests were already performed in [13], but rather to check if the smallest clusters separated from the rest of the algorithm by a barrier of slow interconnect can be of any benefit to the search process. The test conditions were set to amplify possible problems. Only two islands were used in the test, bigger one with 200 individuals and smaller one with 20. The testing program (emulating reconstruction program with Rastrigin formula) introduced only 1 second delay, so the proportion between \(T_f\) and \(T_c\) was unfavourable for the algorithm, compared to the real life case. The results are presented on figure 2.

Migration was performed using GFAL and RFIO on DPM SE. Both islands used MPI for local communication. Figure 2a confirms that the smaller of the islands benefits considerably from turning on the migration. Noticeably better results with the migration turned on, can be also seen on the bigger island, on Figure 2b. These values were achieved with relatively low migration frequency of about 20 migrations per 10000 individuals processed locally. The test confirmed that using even the smallest clusters in the algorithm is beneficial for the quality of the results. The difference in the results diminishes in later generations, which is probably caused by being close to the global optimum at that stage.

9. Test with COMPASS reconstruction program

The test on the real case was performed using CORAL. The optimization concerned 28 parameters concurrently; all of them were geometrical tolerances and were represented by floating point numbers. Each individual was rated on the same set of 200 events. For the sake of the test, the configuration of the algorithm was the same as in the benchmark case. There were two islands, one with 200 slaves and the other with 20, both using MPI for local communication and GFAL for migration with the same
migration intensity. The optimization in each configuration was executed 30 times. This is less than in the test case (because of time constraints), so the graphs are slightly more noisy. The results are presented on figure 3:

**Figure 3.** Values of $\Phi$ of the best individual, normalized to the percent of tracks possible to reconstruct and averaged over 30 runs. Line with red square shows the results from big island with the migration, blue x from big island without the migration, magenta circle from small island with the migration and green * from small island without the migration. Flat orange line at the bottom marks the level set by the default CORAL options (manually optimized). Statistical significance level < 5% is maintained from generation 13 upwards for both islands.

The results show that the optimization of the reconstruction is a problem more difficult, than the multimodal benchmark function used in the test run. The benefits from using the migration are higher and they stay high to the end of the run. The results achieved by the small island without the migration are low and the graph flattens at later generations, which is symptom of being stuck in local optima.

Big differences between the runs with the migration on and off lead to the conclusion that there is still room for improvement with adding more resources. Several tests were performed with 3-island setup (about 500 slaves in total, the same migration intensity) with noticeably better result. The detailed comparison between the default options and the best set found by the algorithm is following:

- reference options: efficiency 79.600% + 20 errors.
- best optimized results: efficiency 84.457% + 19 errors.

**10. Conclusions and future work**

The values achieved by the algorithm, when compared to the default CORAL options, confirm that the reconstruction can be successfully optimized using distributed EA. The overall performance gain over manually optimized parameters is more than 4.8% with 1 error less on the reference sample. It was also confirmed that the optimization of the reconstruction is a multimodal problem and should be treated by heuristic methods. The initial tests with derivative-based method (hill climbing which was added to the program as an operator) show no performance gain at all. Both benchmark and production runs show that the chosen communication model is effective for a broad spectrum of objective function execution times, despite the slow interconnect that is used between the islands. The main strength of the algorithm lies in automatic adaptation of migration frequency and random selection of migration partners. Thanks to these features the migration to local calculations ratio stays the same despite the number of islands used. Therefore the algorithm has no theoretical scaling boundary. However it can be expected that at some threshold the best solution will not have time to spread all over the algorithm until the end of the run. Theoretically this threshold can be estimated using simplified approach presented in [10], where it was shown that in order to reach the takeover condition, the number of epochs should be proportional to $r$ divided by square root of $\delta$. This calculation, although it was done with many simplifying assumptions, shows that the exact barrier should be checked experimentally in the production environment. This makes the analysis of the
behaviour of the algorithm with big number of islands one of the most interesting topics for future work. Unfortunately it was impossible to make this kind of tests in the current setup, because COMPASS VO is a small one and it is supported only by couple of clusters in WLCG. It could be done though if the algorithm is used for a bigger experiment, with more complicated detector and larger computational resources. The drawback of using the method depicted in this paper is that the program optimizes the reconstruction parameters for a particular sample and detector alignment, used for the evaluation. This may result in lack of generality of the optimized parameters and lower performance for different samples and setups. The problem can be corrected, though, with randomization of the sample. The program has partial support for this kind of procedure – every evaluation can use random set of events selected from a larger file. High automation of the program and good results that are achieved even with relatively small resources make it also a good tool for continuous optimization, so the parameters could be optimized for every change in the detector setup. From the point of view of the algorithm itself, it may be useful to replace the $\sigma$ criterion in migration adaptation with some other check that would be more sensitive to the changes on the later stages of population development. There is also some space for performance improvements lying in the implementation of better evolutionary operators and more automation in resources discovery. The work presented here was concentrated on the communication aspects in hybrid EA working in grid environment, so the rest of the EA framework used the most popular, simple algorithms. As there are very comprehensive studies on these topics available, including the classical [8], this would be more of an engineering work than research, but the real application that lies behind this work would surely benefit from it.

Acknowledgements
This work was supported by the EU and MSHE grant nr POIG.02.03.00-00-013/09.

References
[1] Goldberg D 1989 Genetic Algorithms in Search, Optimization & Machine Learning (Addison Wesley, Reading, MA)
[2] Padée A, Kurek K, Zaremba K 2007 Meas. Sci. Technol. 18 2472
[3] The COMPASS Collaboration 1996 Proposal: Common Muon and Proton Apparatus for Structure and Spectroscopy. CERN/SPSLC 96-14, SPSC/P 297
[4] Abbon P et.al. 2007 Nucl. Instr. Meth. A 577 455-518
[5] Kurek K et.al. 2007 Nucl. Instr. Meth. Phys. Res. A 485 720-738
[6] http://wlcg.web.cern.ch/
[7] http://grid.ifca.es/int.eu.grid/
[8] Bäck T, Fogel D, Michalewicz Z. 1996 Handbook of Evolutionary Computation (Oxford University Press)
[9] Tomassini M, Alba E 2002 IEEE Trans. Evol. Comp. 6(5) 443-462
[10] Cantú-Paz E 2000 Efficient and Accurate Parallel Genetic Algorithms (Kluwer Academic Publishers, Norwell, MA)
[11] Munetomo M, Takai Y, Sato Y 1993 An Efficient Migration Scheme for Subpopulation-Based Asynchronously Parallel Genetic Algorithms Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, USA, June 1993) ed S Forrest (Morgan Kaufmann) p 649
[12] Alba E, Troya J 1999 Lecture Notes in Computer Science 1586 248-256
[13] Branke J, Kamper A, Schmeck H 2004 Lecture Notes in Computer Science 3102 923-934
[14] Whitley D 1989 The GENITOR Algorithm and Selective Pressure: Why Rank-Based Allocation of Reproductive Trials is Best Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, USA, June 1989) ed J Schaffer (Morgan Kaufmann) p 116-121