A MapReduce based Glowworm Swarm Optimization Approach for Multimodal Functions

Ibrahim Aljarah and Simone A. Ludwig
Department of Computer Science
North Dakota State University
Fargo, ND, USA
{ibrahim.aljarah,simone.ludwig}@ndsu.edu

***Abstract*—In optimization problems, such as highly multimodal functions, many iterations involving complex function evaluations are required. Glowworm Swarm Optimization (GSO) has to be parallelized for such functions when large populations capturing the complete function space, are used. However, large-scale parallel algorithms must communicate efficiently, involve load balancing across all available computer nodes, and resolve parallelization problems such as the failure of nodes. In this paper, we outline how GSO can be modeled based on the MapReduce parallel programming model. We describe MapReduce and present how GSO can be naturally expressed in this model, without having to explicitly handle the parallelization details. We use highly multimodal benchmark functions for evaluating our MR-GSO algorithm. Furthermore, we demonstrate that MR-GSO is appropriate for optimizing difficult evaluation functions, and show that high function peak capture rates are achieved. We show with the experiments that adding more nodes would help to solve larger problems without any modifications to the algorithm structure.**

***Keywords*—Parallel Processing, Optimization, MapReduce, Hadoop**

I. Introduction

Swarm intelligence [1] simulates the natural swarms such as ant colonies, flocks of birds, bacterial growth, and schools of fishes. The behavior of the swarm is based on the sense of the member's interactions in the swarm by exchanging the local information with each other to help reaching the food sources. There is no central member in the swarm, but rather all swarm members participate equally to achieve the goal.

Glowworm Swarm Optimization (GSO) [2] is an optimization algorithm, which belongs to the swarm intelligence field [1] that is inspired by simulated experiments of the behavior of insects that are called glowworms or lighting worms. These glowworms are able to control their light emission and use it to glow for different objectives such as e.g., attracting the other worms during the breeding season. Most swarm intelligence algorithms are concerned with locating the global solution based on the objective function for the given optimization problem. In addition, locating one global solution is considered easier than locating multiple solutions. The GSO algorithm is especially useful for a simultaneous search of multiple solutions, having different or equal objective function values. To achieve this target, a swarm must have the ability to divide itself into separated groups.

The GSO algorithm is one of the recent swarm intelligence algorithms that have been used in many applications such as the hazard sensing in ubiquitous environments [3], mobile sensor network and robotics [2], and data clustering [4], because of its implementation simplicity and the need to tune a small number of parameters [2, 3, 5]. Some functions such as multimodal functions are functions with many local maxima also referred to as peaks. Multimodal function optimization is not aiming to find the global maximum only, but rather all maxima based on some constraints. The peak count increases for high dimensional spaces, therefore, each function evaluation requires a long time to compute in order to find optimal target peaks at the end of the optimization process.

To optimize such functions, the number of individuals must be increased to share more local information for locating more peaks. To solve the high computation time in these situations, the algorithm must be parallelized in an efficient way to find the maxima in an acceptable amount of time.

Parallel algorithms suffer from a wide range of problems such as inefficient communication, or unfair load balancing, which makes the process of scaling the algorithms to large numbers of processors very difficult. Also, node failure affects the parallel algorithms, thus, reduce the algorithm's scalability. Therefore, any parallel algorithm developed should handle large amounts of data and scale well by increasing the compute nodes while maintaining high quality results.

The MapReduce programming model [6], developed by Google, has recently become very promising model for parallel processing in comparison to the message passing interface parallelization technique [7]. The strength that makes the MapReduce methodology to be good model for parallelizing the tasks is that the process can be performed automatically without the knowledge of parallel programming. In addition,the MapReduce methodology provides fault-tolerance, load balancing, and data locality.

Besides Google's MapReduce implementation, there are other MapReduce open source implementations available such as Apache Hadoop MapReduce [8], and Disco [9]. MapReduce is a highly scalable model and most applicable when the task considered is a data intensive task. MapReduce is suggested when computing resources have restrictions on multiprocess-

ing and large shared-memory hardware. MapReduce has also been adopted by many companies in industry (e.g., Facebook [10], and Yahoo [11]). In academia, researchers benefit from MapReduce for scientific computing, such as in the areas of Bioinformatics [12] and Geosciences [13] where codes are written as MapReduce programs.

In MapReduce, the problem is formulated as a functional procedure using two core functions: the *Map* function and *Reduce* function. The main idea behind the MapReduce model is the mapping of data into a list of <key,value> pairs, and then applying the reducing operation over all pairs with the same key. The *Map* function iterates over a large number of input units and processes them to extract intermediate output from each input unit, and all output values that have the same key are sent to the same *Reduce* function. On the other hand, the *Reduce* function collects the intermediate results with the same key that is retrieved by the *Map* function, and then generates the final results. Figure 1 shows the MapReduce's core functions.

Map Operation:
Map (k, v) $\rightarrow$ [(k', v')]

Reduce Operation:
Reduce (k', [v']) $\rightarrow$ [(k', v')]

Fig. 1. The *Map* and *Reduce* Operations

Apache Hadoop [8] is an implementation that uses the MapReduce methodology, which was developed in order to effectively deal with massive amounts of data or data-intensive applications. One of the strengths of Hadoop is its scalability. It works with one machine, and can grow quickly to thousands of computer nodes, which is developed to run on commodity hardware. Apache Hadoop consists of two main components: Hadoop Distributed File System (HDFS), which is used for data storage, and MapReduce, which is used for data processing. HDFS provides a high-throughput access to the data while maintaining fault tolerance to avoid the failure node issues by replicating multiple copies of data blocks. MapReduce works together with HDFS to provide the ability to move the computation to the data to maintain the data locality feature. Figure 2 shows the Hadoop architecture diagram and control flow between the two components. Interested readers may refer to [8] for more details. The Hadoop framework is used in our proposed algorithm implementation.

In this paper, the MapReduce methodology is utilized to create a parallel glowworm swarm optimization algorithm. The purpose of applying MapReduce to glowworm swarm optimization goes further than merely being a hardware utilization. Rather, a distributed model is developed, which achieves better solutions since it is scalable with a reduced overall computation time.

This paper presents a parallel Glowworm swarm optimization (MR-GSO) algorithm based on the MapReduce framework. In this work, we have made the following key contributions:

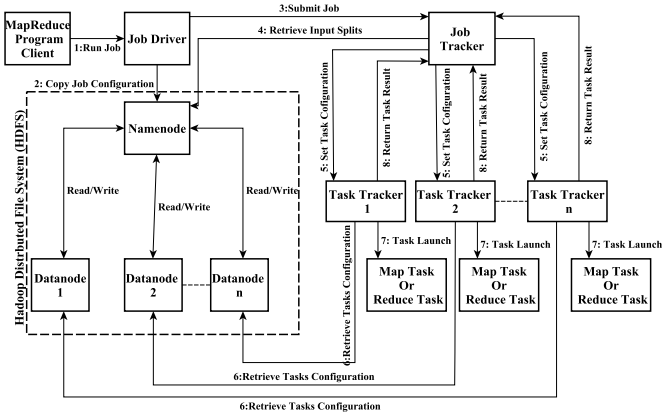


Fig. 2. Hadoop Architecture Diagram

1) The proposed algorithm (MR-GSO) makes use of the MapReduce framework that has been proven successful as a parallelization methodology.
2) The proposed algorithm has been tested on large scale multimodal functions with different dimensions to show the speedup and scalability while maintaining the optimization quality.

The remainder of this paper is organized as follows: Section 2 presents the related work in the area of parallel optimization algorithms. In Section 3, the glowworm swarm optimization approach is introduced as well as our proposed MR-GSO algorithm. Section 4 presents the experimental evaluation, and Section 5 presents our conclusions.

II. Related Work

The parallelization of optimization algorithms has received much attention to reduce the run time for solving large-scale problems [14, 15]. Parallel algorithms make use of multiple processing nodes in order to achieve a speedup as compared to running the sequential version of the algorithm on only one processor [16]. Many parallel algorithms have been proposed to meet the difficulties of implementing optimization algorithms.

Many of the existing algorithms in the literature apply the message passing methodology (MPI) [7]. In [15], a parallel genetic algorithm was proposed using the MPI library on a Beowulf Linux Cluster with the master slave paradigm. In [14], an MPI based parallel particle swarm optimization algorithm was introduced. However, MPI is not the best choice for parallelization because of the weakness of having to handle the failure of nodes.

MapReduce [6] is easier to understand, while MPI [7] is somehow more complicated since it has many instructions. MPI can reuse parallel processes on a finer granularity level. MapReduce communicates between the nodes by disk operations (the shared data is stored in distributed file system such as HDFS), which is faster than local file systems, while MPI communicates via the message passing model. MapReduce provides fault-tolerance of node failures, while the MPI processes are terminated when a node fails.

In [17], MRPSO incorporated the MapReduce model to parallelize particle swarm optimization by applying it on computationally data intensive tasks. The authors presented a radial basis function as the benchmark for evaluating their MRPSO approach, and verified that MRPSO is a good approach for optimizing data-intensive functions.

In [18], the authors made an extension of the genetic algorithm with the MapReduce model, and successfully proved that the genetic algorithm can be parallelized easier with the MapReduce methodology. In [19], the authors proposed a MapReduce based ant colony approach. They show how ant colony optimization can be modeled with the MapReduce framework. They designed and implemented their algorithm using Hadoop.

Comparing our proposed algorithm to the algorithms listed above, all MapReduce implementations were used to optimize single objective functions, whereas in our proposed algorithm, the algorithm searches for multiple maxima for difficult multimodal functions. To the best of our knowledge, MR-GSO is the first work on the parallelization of glowworm swarm optimization. Furthermore, it is the first work using the MapReduce methodology, which is considered an alternative model for parallel processing over the message passing methodology [7]. GSO can be naturally expressed with the MapReduce methodology, and therefore, easily be parallelized in order to be able to solve computationally expensive multimodal functions with high dimensionality.

III. Proposed Approach

A. Introduction to Glowworm Swarm Optimization

Glowworm swarm optimization (GSO) is a swarm intelligence method first introduced by Krishnan and Ghose in 2005 [2]. The swarm in the GSO algorithm is composed of N individuals called glowworms. A glowworm i has a position $X_i(t)$ at time t in the function search space, a light emission which is called a luciferin level $L_i(t)$, and a local decision range $rd_i(t)$. The luciferin level is associated with the objective value of the individual's position based on the objective function J.

A glowworm that emits more light (high luciferin level) means that it is closer to an actual position and has a high objective function value. A glowworm is attracted by other glowworms whose luciferin level is higher than its own within the local decision range. If the glowworm finds some neighbors with a higher luciferin level and within its local range, the glowworm moves towards them. At the end of the process, most glowworms will be gathered at the multiple peak locations in the search space.

The GSO algorithm consists of four main stages: glowworm initialization, luciferin level update, glowworm movement, and glowworm local decision range update.

In the first stage, N glowworms are randomly deployed in the specific objective function search space. In addition, in this stage the constants that are used for the optimization are initialized, and all glowworms luciferin levels are initialized with the same value (L_0). Furthermore, local decision range rd and radial sensor range r_s are initialized with the same initial value (r_0).

The luciferin level update is considered the most important step in glowworm optimization because in this stage the objective function is evaluated at the current glowworm position (X_i). The luciferin level for all swarm members are modified according to the objective function values. The process for the luciferin level update is done with the following equation:

$$L_i(t) = (1-\rho)L_i(t-1) + \gamma J(X_i(t)) \tag{1}$$

where $L_i(t)$ and $L_i(t-1)$ are the new luciferin level and the previous luciferin level for glowworm i, respectively, ρ is the luciferin decay constant ($\rho \in (0,1)$), γ is the luciferin enhancement fraction, and $J(X_i(t))$ represents the objective function value for glowworm i at current glowworm position (X_i) at iteration t.

After that and throughout the movement stage, each glowworm i tries to extract the neighbor group $N_i(t)$ based on the luciferin levels and the local decision range (r_d) using the following rule:

$$j \in N_i(t) \;\; \textit{iff} \;\; d_{ij} < rd_i(t) \;\; \textit{and} \;\; L_j(t) > L_i(t) \tag{2}$$

where j is one of the glowworms close to glowworm i, $N_i(t)$ is the neighbor group, d_{ij} is the Euclidean distance between glowworm i and glowworm j, $rd_i(t)$ is the local decision range for glowworm i, and $L_j(t)$ and $L_i(t)$ are the luciferin levels for glowworm j and i, respectively.

After that, the actual selected neighbor is identified by two operations: the probability calculation operation to figure out the movement direction toward the neighbor with the higher luciferin value. This is done by applying the following equation:

$$Prob_{ij} = \frac{L_j(t) - L_i(t)}{\sum_{k \in N_i(t)} L_k(t) - L_i(t)} \tag{3}$$

where j is one of the neighbor group $N_i(t)$ of glowworm i.

After the probability calculation, in the second operation, glowworm i selects a glowworm from the neighbor group using the roulette wheel method whereby the higher probability glowworm has more chance to be selected from the neighbor group.

Then, at the end of the glowworm movement stage, the position of the glowworm is modified based on the selected neighbor position using the following equation:

$$X_i(t) = X_i(t-1) + s \; \frac{X_j(t) - X_i(t)}{\delta_{ij}} \tag{4}$$

where $X_i(t)$ and $X_i(t-1)$ are the new position and previous position for the glowworm i, respectively, s is a step size constant, and δ_{ij} is the Euclidean Distance between glowworm i and glowworm j.

The last stage of GSO, is the local decision range update, where the local decision range rd_i is updated in order to add flexibility to the glowworm to formulate the neighbor group

in the next iteration. The following equation is used to update rd_i in the next iteration:

$$rd_i(t) = min\{rs, max[0, rd_i(t-1) + \beta(nt - |N_i(t-1)|)]\} \quad (5)$$

where $rd_i(t)$ and $rd_i(t-1)$ are the new local decision range, and the previous local decision range for glowworm i respectively, rs is the constant radial sensor range, β is a model constant, nt is a constant parameter used to control the neighbor count, and $|N_i(t)|$ is the actual number of neighbors.

B. Proposed MapReduce GSO Algorithm (MR-GSO)

The grouping nature of glowworm swam optimization makes it an ideal candidate for parallelization. Based on the sequential procedure of glowworm optimization discussed in the previous section, we can employ the MapReduce model. The MR-GSO consists of two main phases: Initialization phase, and MapReduce phase.

In the initialization phase, an initial glowworm swarm is created. For each glowworm i, a random position vector (X_i) is generated using uniform randomization within the given search space. Then, the objective function J is evaluated using the X_i vector. After that, the luciferin level (L_i) is calculated by Equation 1 using the initial luciferin level L_0, $J(X_i)$, and other given constants. The local decision range r_d is given an initial range r_0. After the swarm is updated with this information, the glowworms are stored in a file on the distributed file system as a <Key,Value> pair structure, where Key is a unique glowworm ID i and Value is the glowworm information. The initial stored file is used as input for the first MapReduce job in the MapReduce phase.

The representation structure of the <Key,Value> pairs are used in the MR-GSO algorithm as shown in Figure 3. The main glowworm components are delimited by semicolon, while the position X_i vector component is delimited by comma, where m is the number of dimensions used. In the

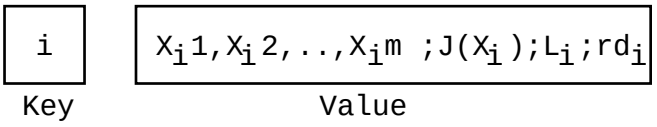


Fig. 3. Glowworm Representation Structure

second phase of MR-GSO, an iterative process of MapReduce jobs is performed where each MapReduce job represents an iteration in the glowworm swarm optimization. The result of each MapReduce job is an updated glowworm swarm with updated information, which is then used as the input for the next MapReduce job.

In each MapReduce job, the algorithm focuses on the time consuming stages of the luciferin level update and glowworm movement to benefit from the power of the MapReduce model. In the movement stage, each glowworm i extracts the neighbor group $N_i(t)$ based on Equation 2, which requires distance calculations and luciferin level comparisons between each glowworm and other swarm members to locate the neighbor group. This process is executed N^2 times, where N is the swarm size. The neighbor group finding process is accomplished by the *Map* function that is part of a MapReduce job.

Before the neighbor group finding process is done in the *Map* function, a copy of the stored glowworm swarm ($TempSwarm$) is retrieved from the distributed file system, which is a feature provided by the MapReduce framework for storing files. In addition, the other information such as the GSO constants s, ρ, γ, β, nt, and rs that are used in the GSO movement equations, are retrieved from the job configuration file.

After that, the neighbor group finding process is started when the *Map* function receives <Key,Value> pairs from the MapReduce job driver, where Key is the glowworm ID i and the Value is the glowworm information. However, the *Map* function processes the Value by breaking it into the main glowworm components (X_i, $J(X_i)$, L_i, and rd_i), which are used inside the *Map* function. Then, a local iterative search is performed on $TempSwarm$ to locate the neighbor group using Equation 2. After that, the neighbor probability values are calculated based on Equation 3 to find the best neighbor using the roulette wheel selection method. At the end of the *Map* operation, the *Map* function emits the glowworm ID i with its Value and glowworm ID i with the selected neighbor position vector (X_j) to the *Reduce* function. The *Map* function works as shown in Figure 4 outlining the pseudo code of the *Map* operation.

As an intermediate step in the MapReduce job, the emited intermediate output from the mapper function is partitioned using the default partitioner by assigning the glowworms to the reducers based on their IDs using the modulus hash function.

The *Reduce* function in the MapReduce job is responsible for updating the luciferin level L_i which is considered the most expensive step in the glowworm optimization, since in this stage the objective function is evaluated for the new glowworm position. The luciferin level updating process is started when the *Reduce* function receives <Key,ListofValues> pairs from the *Map* function where Key is the glowworm ID and the $ListofValues$ contains the glowworm value itself and its best neighbor position (X_j). The reduce function extracts the neighbor position vector (X_j) and glowworm information (X_i, $J(X_i)$, L_i, and rd_i). Then, the updating of the glowworm position vector is done using Equation 4. After that, the objective function is evaluated using the new glowworm position vector, and then the luciferin level is updated using Equation 1. Also, rd_i is updated using Equation 5. At the end, the *Reduce* function emits the glowworm ID i with new updated glowworm information. The pseudo-code of the *Reduce* function is shown in Figure 5.

At the end of the MapReduce job, the new glowworm swarm replaces the previous swarm in the distributed file system, which is used by the next MapReduce job.

function *Map* (Key: GlowwormID, Value: Glowworm)
Initialization:
$glowwormID$=Key
$glowwormValue$=$Value$
$i = glowwormID$
//Extract the information from the Glowworm
extractInfo(X_i,J_i,L_i,rd_i)
//Read the copy from the glowworm
swarm from the Distributed Cache
read($TempSwarm$)
for each $glowworm\ j$ in $TempSwarm$
X_j=extractPosition($glowworm$)
L_j=extractluciferin($glowworm$)
$EDist$=returnEDistance(X_i,X_j)
if ($EDist < rd_i$ and $L_j > L_i$) **then**
$NeighborsGroup.add(j)$
end if
end for
if ($NeighborsGroup.size() < 0$) **then**
for each $glowworm\ j$ in $NeighborsGroup$ **do**
//calculate the probabilities from
the NeighborsGroup using Equation 2
prob[j]=calculateProbability(i,j)
end for
end if
nj=selectBestNeighbor(prob) //using the roulette wheel
X_j=extractPosition(nj)
$newValuenb$=createValue($NeighborsGroup.size()$,X_j)
Emit($glowwormID$, $newValuenb$)
Emit($glowwormID$, $glowwormValue$)
end function

Fig. 4. Algorithm - Map Function

function *Reduce* (Key:glowwormID,ValList)
$glowwormID$=Key
$i = glowwormID$
for each $Value$ in ValList
if ($Value$ is the Neighbor case) **then**
//Extract the Neighbor information from the Value
extractInfo(X_j)
//Extract the nbSize from the Value
extractInfo($nbSize$)
else
//Extract the information from the current glowworm
$glowworm_i$=NULL
extractInfo(X_i,J_i,L_i,rd_i)
fill($glowworm_i$,X_i,Jx_i,L_i,rd_i)
end if
end for
//calculate the new position for glowworm i
using Equation 4
$newX$=$calculateNewX$(X_i,X_j)
//update luciferin level for glowworm i
using objective function formula J
$newJx$=$calculateNewJx$($newX$)
//update luciferin level for glowworm i
using Equation 1
$newL$=$calculateNewX$(L_i,$newJx$)
//calculate the new rd for glowworm i
using Equation 5
$newrd$=$calculateNewrd$(rd_i,$nbSize$)
$glowworm_i$.update($newX$,$newJx$,$newL$,$newrd$)
Emit($glowwormID$, $glowworm_i$)
end function

Fig. 5. Algorithm - Reduce Function

IV. Experiments and Results

In this section, we describe the optimization quality and discuss the running time of the measurements for our proposed algorithm. We focus on scalability in terms of speedup and the optimization quality.

A. Environment

We ran the MR-GSO experiments on the NDSU[2] Hadoop cluster. The NDSU Hadoop cluster consists of only 18 nodes containing 6GB of RAM, 4 Intel cores (2.67GHz each) with HDFS 2.86 TB aggregated capacity. For our experiments, we used Hadoop version 0.20 (new API) for the MapReduce framework, and Java runtime 1.6 to implement the MR-GSO algorithm.

B. Benchmark Functions

To evaluate our MR-GSO algorithm, we used three standard multimodal benchmark functions. The benchmark functions that are used are the following:

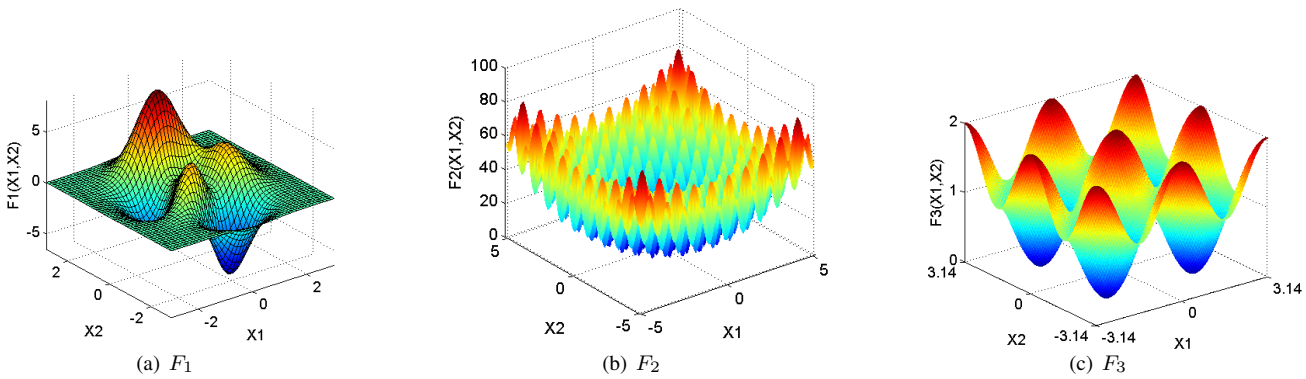
[2]http://www.ndsu.edu

- F_1: The Peaks function in Equation 6 is a function of two variables, obtained by translating and scaling Gaussian distributions. It has multiple peaks which are located at (0,1.58), (0.46,0.63), and (1.28,0) with different peak function values. The function has the following definition:

$$\begin{aligned} F_1(X_1, X_2) = 3(1 - X_1)^2 e^{-[{X_1}^2 + (X_2+1)^2]} \\ -10(\frac{X_1}{5} - X_1^3 - X_2^5) e^{-[X_1^2 + X_2^2]} \\ -(\frac{1}{3}) e^{-[(X_1+1)^2 + X_2^2]} \end{aligned} \quad (6)$$

  Figure 6(a) shows the Peaks function visualized for two dimensions.

- F_2: The Rastrigin function is a highly multimodal function with the locations of the minima and maxima regularly distributed. This function presents a fairly difficult problem due to its large search space and its large number of local minima and maxima. We restricted the function to the hypercube $-5.12 \leq X_i \leq 5.12, i = 1, ..., m$. The function has 100 peaks for 2 dimensions within the given



(a) F_1 (b) F_2 (c) F_3

Fig. 6. Benchmark Functions. 6(a) F_1: Peaks Function. 6(b) F_2: Rastrigin Function. 6(c) F_3: Equal-peaks-A Function.

range. The function has the following definition:

$$F2(X_i) = 2m + \sum_{i=1}^{m}[{X_i}^2 - 10\cos(2X_i)] \quad (7)$$

Figure 6(b) shows the Rastrigin function visualized for two dimensions.

- F_3: The Equal-peaks-A function is a highly multimodal function in the m-dimensional search space. All local maxima of the Equal-peaks-A function have equal function values. The function search space ($-\pi \leq X_i \leq \pi$) is used, where, i = 1,...,m. The function has 3^m peaks such as for m=2 dimensions, the function has 9 peaks within the given range. We use the function F_3 to test higher dimensional search spaces such as m = 2, 3 and 4. The function has the following definition:

$$F_3(X_i) = \sum_{i=1}^{m}[\cos^2(X_i)] \quad (8)$$

Figure 6(c) shows the Equal-peaks-A function visualized for two dimensions.

C. Evaluation Measures

In our experiments, we used the parallel Speedup [16] measure to evaluate the performance of our MR-GSO algorithm, which is calculated using the following equation:

$$Speedup = \frac{T_2}{T_n} \quad (9)$$

where T_2 is the running time using 2 nodes, and T_n is the running time using n nodes, where n is a multiple of 2.

The speedup is obtained by fixing the swarm size while increasing the number of cluster nodes to evaluate the algorithm's ability to scale with increasing numbers of the cluster nodes. For the optimization quality, we use the Peaks Captured Rate (PCR) and the average minimum distance to the peak locations (D_{avg}) [20]. The peak is considered captured if there are three glowworms near it with distance less than or equal ϵ. In this paper, we used the distance ϵ = 0.05 recommended in [20].

PCR is given by the following equation:

$$PCR = \frac{Number\ of\ Peaks\ Captured}{Number\ of\ All\ Peaks} \times 100\% \quad (10)$$

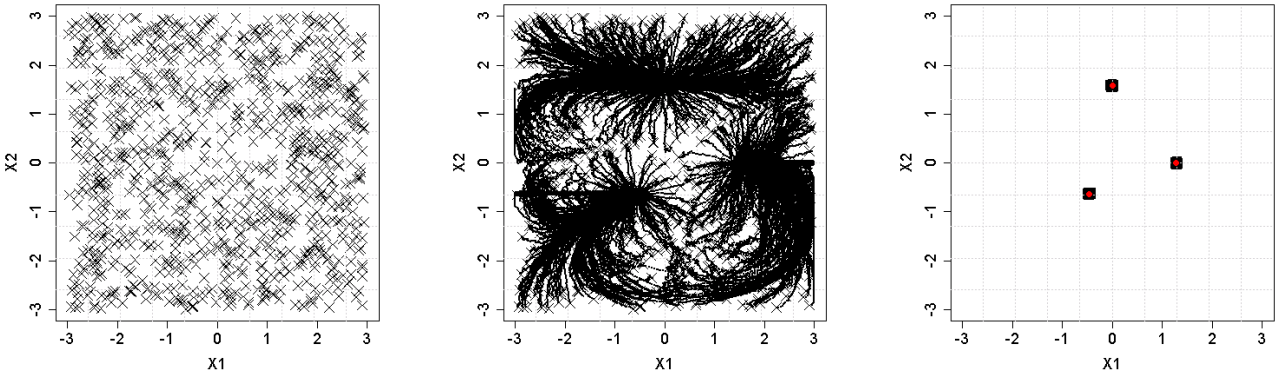
The average minimum distance to the peak locations D_{avg} is given by the following equation:

$$D_{avg} = \frac{1}{N} \times \sum_{i=1}^{N} \min_{\{1 \leq j \leq Q\}} \{\delta_{i1} ... \delta_{iQ}\} \quad (11)$$

where δ_{ij} is the Euclidean Distance between the location of glowworm X_i and S_j; X_i and S_j are the locations of glowworm i and peak j, respectively, and Q is the number of available peak locations; N is the number of glowworms in the swarm.

The best result for these measures will have a high PCR and low D_{avg}. For example, if we get a low D_{avg} and a low PCR, this means that the glowworms gathered in only a few peaks, and did not capture the other peaks. A high PCR, close to 100%, means that MR-GSO captured most of the peaks, whereas a low D_{avg}, close to zero, implies that all glowworms are close to the peaks, and thus, this ensures a gathering of the glowworms at the peak locations.

We used the GSO settings that are recommended in [5]. We used the luciferin decay constant $\rho = 0.4$; the luciferin enhancement constant $\gamma = 0.6$; the constant parameter $\beta = 0.08$; the parameter used to control the number of neighbors $nt = 5$; the initial luciferin rate $L_0 = 5.0$; the step size $s = 0.03$. In addition, the local decision range r_d and the radial sensor range r_s are problem based values. In our experiments, the local decision range rd is kept constant (rs=rd=r_0) throughout the optimization process. Preliminary experiments were done to decide whether to use an adaptive or constant rd. The constant rd achieved better results, since it ensures that the glowworm moves even if it has many neighbors. If there are many neighbors around, then the glowworm keeps moving towards the peaks, unlike the adaptive rd, where if the glowworm has many neighbors, it does not move and therefore, the new rd is 0 based on Equation 5. The best r_0 values for the given benchmark functions are chosen based on preliminary experiments.



(a) Initial State (b) Glowworms Movements (c) Glowworms Final Locations with Peaks Locations

Fig. 7. Optimization process for the peaks function (F1) with swarm size=1, 000, number of iterations=200, and r_0=1.0: the glowworms start from an initial random location and move to one of the function peaks. 7(a) The initial random glowworm locations. 7(b) The movements of the glowworms throughout the optimization process. 7(c) The final locations of glowworms (small squares) after the optimization process with the peak locations (red solid circles).

D. Results

To evaluate the MR-GSO algorithm, the experiments are done measuring the PCR, D_{avg}, running time, and speedup for the mentioned benchmarks.

Figure 7 shows the MR-GSO optimization quality results visualized for the $F1$ benchmark for two dimensions. Figure 7(a) shows the initial state of the glowworm swarm distributed in the search space using the random uniform distribution. The second part (Figure 7(b)) shows the movement of each of the glowworms in Figure 7(a) moving towards the closest peak. At the end, all glowworms are gathered at the 3 peaks as shown in Figure 7(c). For this function, the results show that the MR-GSO algorithm is able to locate all 3 peaks, and therefore, achieves a PCR of 100%. In addition, the average minimum distance for the glowworms is very good with $D_{avg} = 0.0193$, which is fairly close to zero.

Figure 8 shows the MR-GSO optimization quality results visualized for the $F2$ benchmark for two dimensions. The results show that the MR-GSO algorithm is able to locate 96 out of 100 peaks (PCR=96%). Furthermore, D_{avg} is very low with a value of 0.031.

The results for the $F3$ benchmark with two dimensions are given in Figure 9. The MR-GSO algorithm is able to locate all 9 peaks (PCR=100%) for this function. Also, a good value of D_{avg} is obtained with a low value equal to 0.017.

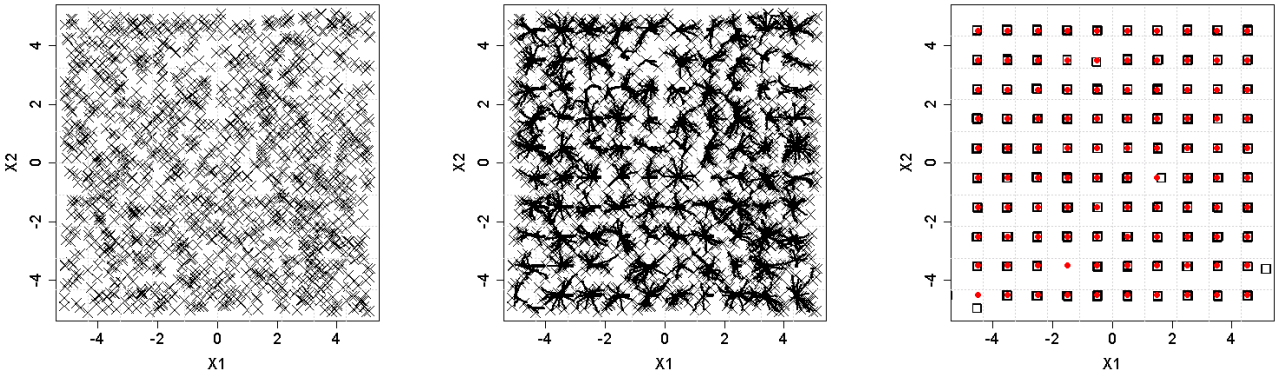
The optimization quality results for the $F3$ function with three dimensions are shown in Figure 10. The PCR and D_{avg} for each iteration using different numbers of swarm sizes (starting from 10,000 to 100,000) are presented. As can be noted from Figure 10(a), the PCR is improving for increasing swarm sizes. In addition, the number of iterations needed to capture all peaks is reduced, such as, the PCR converges to 100% with a swarm size of 10,000 at iteration 73, while with a swarm size of 100,000 the PCR converges at iteration 36. Also, Figure 10(b) shows that the average minimum distance is improved when the swarm size is increased while maintaining low values for all swarm sizes. Figure 10(c) visualizes an example of the final glowworm location with peak locations for a swarm size of 30,000, where the PCR is 100% and D_{avg} is 0.0186.

The optimization quality results for the $F3$ function with four dimensions are shown in Figure 11. Figure 11(a) clarifies the impact of the swarm size on the PCR. However, 200 iterations capture 65% of the peaks with a swarm size of 10,000, while with a swarm size of 100,000 the PCR converges to 100% at iteration 123. Also, Figure 11(b), using the log scale for the y axis, shows that larger swarm sizes give better average minimum distance results maintaining low values for all swarm sizes.

We ran MR-GSO with 18 cluster nodes by increasing the number of nodes in each run by multiples of 2. In each run, we report the running time and speedup (average of 25 iterations) of MR-GSO. The running times and speedup measures are shown in Figure 12. Figures 12(a), 12(b) and 12(c) show the running times for the 3 swarm sizes of 100,000, 200,000, and 300,000 glowworms. As can be seen by all subfigures, the running time improves faster at the beginning than the end when increasing the number of nodes. Furthermore, the impact of the swarm size on the running time is well observed. Running the algorithms on 2 nodes takes 550, 2170, and 3335 seconds for 100,000, 200,000, and 300,000 glowworms, respectively.

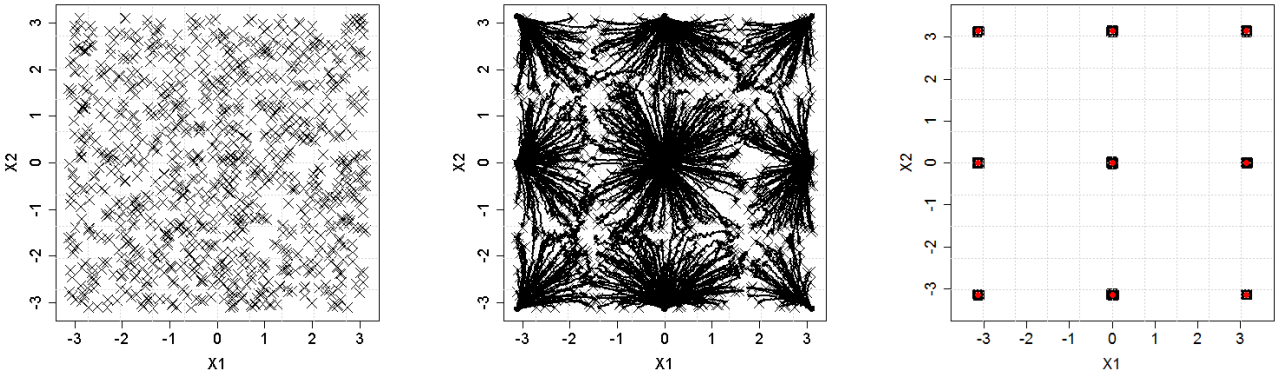
In Figures 12(d), 12(e) and 12(f), the speedup results using different swam sizes with different numbers of nodes are shown, highlighting the scalability of the algorithm. As can be inferred from the figures, the speedup for N=100,000 was very close to the linear speedup (optimal scaling) using 4, 6, and 8 nodes. It diverges from the linear speedup because of the overhead of the Hadoop framework, which results from the management of starting MapReduce jobs, starting mappers and reducers operations, and serializing/deserializing intermediate outputs, and storing the outputs to the distributed file system.

The same behavior is observed for N=200,000 and N=300,000. For N=200,000 the speedup is very close to the



(a) Initial State (b) Glowworms Movements (c) Glowworms Final Locations with Peaks Locations

Fig. 8. Optimization process for the Rastrigins function (F2) with swarm size=$1,000$, number of iterations=200, and r_0=0.5: the glowworms start from an initial random location and move to one of the function peaks. 8(a) The initial random glowworm locations. 8(b) The movements of the glowworms throughout the optimization process. 8(c) The final locations of glowworms (small squares) after the optimization process with the peak locations (red solid circles).



(a) Initial State (b) Glowworms Movements (c) Glowworms Final Locations with Peaks Locations

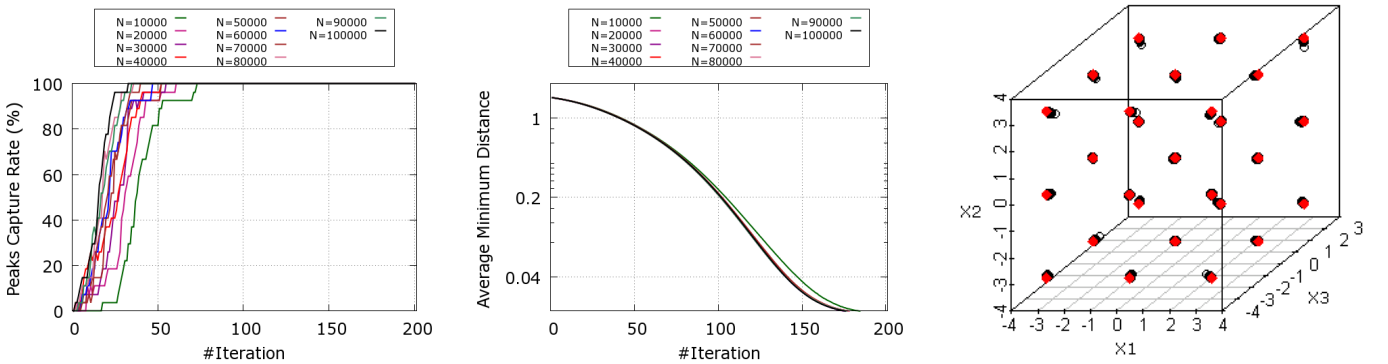
Fig. 9. Optimization process for the Equal-peaks-A function (F3) with swarm size=$1,500$, number of iterations=200, and r_0=1.5: the glowworms start from an initial random location and move to one of the function peaks. 9(a) The initial random glowworm locations. 9(b) The movements of the glowworms through the optimization process. 9(c) The final locations of glowworms (small squares) after the optimization process with the peak locations (red solid circles) for swarm size 30,000.

linear one using 2 to 12 nodes, but then it diverges from the optimal line with a smaller difference compared to N=100,000. For N=300,000, the speedup is close to the linear one with 12 nodes, then it starts to have a larger difference for larger numbers of nodes, but comparing this difference with the one using N=200,000 and N=100,000 is much smaller. Therefore, we can conclude that the overhead of the Hadoop framework can be avoided when using larger numbers of swarm sizes, and thus the speedup is closer to the optimal one. In addition, the improvement factor of MR-GSO's running times for the swarm sizes of N=100,000, N=200,000 and N=300,000 are 4.95, 6.93, 7.41 respectively, compared to the running time using 2 nodes.

V. Conclusion

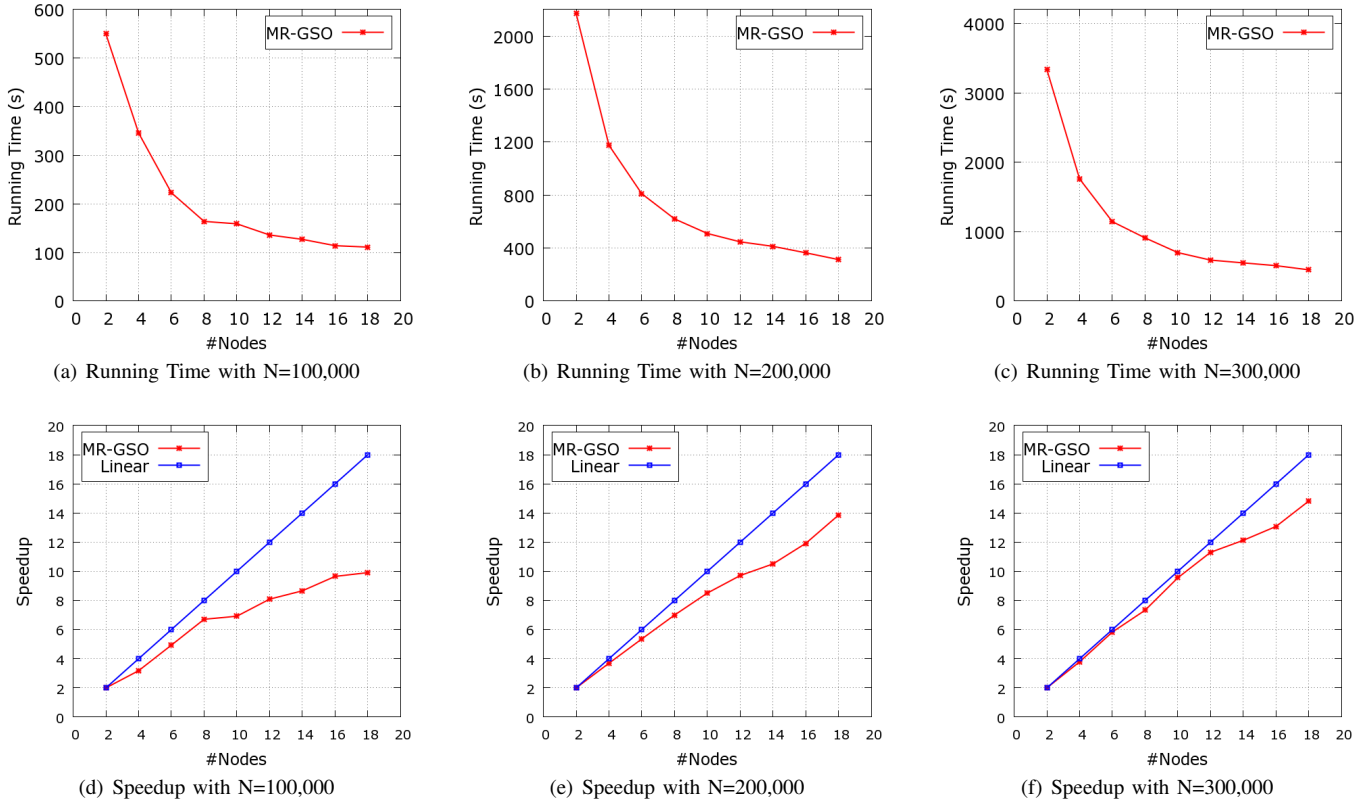
In this paper, we proposed a scalable MR-GSO algorithm using the MapReduce parallel methodology to overcome the computational inefficiency of glowworm swarm optimization when difficult multimodal functions are to be optimized. Since large-scale parallel algorithms must communicate efficiently, balance the load across the available computer nodes, and resolve parallelization problems such as the failure of nodes, MapReduce was chosen since it provides all these features. We have shown that the glowworm swarm optimization algorithm can be successfully parallelized with the MapReduce methodology.

Experiments were conducted with three multimodal functions in order to measure the peak capture rate, average minimum distance to the peak locations, and the speedup of our algorithm. The peak capture rates obtained as well as the average minimum distance values for the three benchmark functions are higher than the ones provided in previous experiments conducted without a parallel framework. This shows the benefit of the parallelization effect on the solution quality. Adding more glowworms reduces the number of iterations required and leads to an overall improvement in optimization quality. In addition, the scalability analysis revealed that MR-GSO scales very well with increasing swarm sizes, and scales



(a) Peaks Capture Rate (b) Average Minimum Distance (c) Glowworms Final Locations with Peaks Locations

Fig. 10. Optimization process for the Equal-peaks-A function (F3) using 3 dimensions with 200 iterations, and r_0=2.0. 10(a) The Peaks capture rate for increasing swarm sizes. 10(b) The average minimum distance for increasing swarm sizes. 10(c) The final locations of glowworms (small squares) after the optimization process with peak locations (red solid circles).



(a) Running Time with N=100,000 (b) Running Time with N=200,000 (c) Running Time with N=300,000

(d) Speedup with N=100,000 (e) Speedup with N=200,000 (f) Speedup with N=300,000

Fig. 12. The Running time and Speedup results for the Equal-peaks-A function (F3) with 4 dimensions. 12(a), 12(b) and 12(c): The Running time with N=100,000, N=200,000 and N=300,000, respectively. 12(d), 12(e) and 12(f): The Speedup with N=100,000, N=200,000 and N=300,000, respectively.

very close to the linear speedup while maintaining optimization quality.

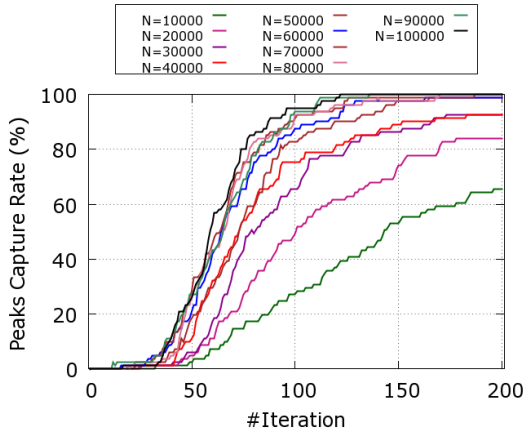
Our future plan is to include measurements for multimodal functions with larger search spaces and higher dimensions as well as using larger swarm sizes. Also, we will investigate the impact of the GSO settings on the optimization quality. Furthermore, we will apply the MR-GSO algorithm on practical applications such as data clustering.

REFERENCES

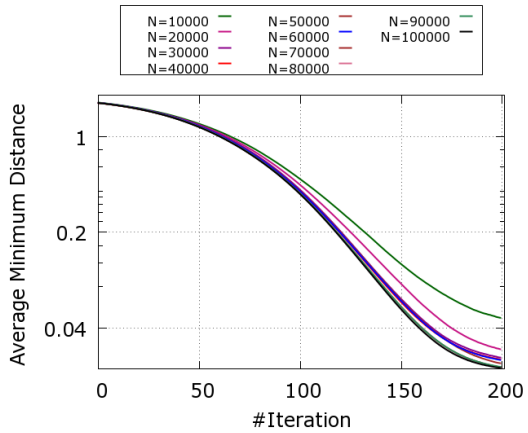
[1] J. Han, *Computational Intelligence: An Introduction 2nd Edition*. Wiley, 2007.

[2] K. Krishnanand and D. Ghose, "Detection of multiple source locations using a glowworm metaphor with applications to collective robotics," in *IEEE Swarm Intelligence Symposium*, Pasadena, CA, USA, june 2005, pp. 84 – 91.

[3] K. N. Krishnanand and D. Ghose, "Glowworm swarm optimization algorithm for hazard sensing in ubiquitous



(a) Peaks Capture Rate



(b) Average Minimum Distance

Fig. 11. Optimization process for the Equal-peaks-A function (F3) using 4 dimensions with 200 iterations, and r_0=2.0. 11(a) The Peaks capture rate. 11(b) The average minimum distance.

environments using heterogeneous agent swarms," *Soft Computing Applications in Industry*, vol. 226, pp. 165–187, 2008.

[4] Y. Z. Zhengxin Huang, "Using glowworm swarm optimization algorithm for clustering analysis," *Journal of Convergence Information Technology*, 2011.

[5] K. Krishnanand and D. Ghose, "Glowworm swarm optimisation: a new method for optimising multi-modal functions," *International Journal of Computational Intelligence Studies*, vol. 1, pp. 93–119, 2009. [Online]. Available: http://inderscience.metapress.com/content/H87J31211158L162

[6] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," 2004, pp. 137–150.

[7] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference*. MIT Press Cambridge, MA, USA, 1995.

[8] (2011) Apache software foundation, hadoop mapreduce. [Online]. Available: http://hadoop.apache.org/mapreduce

[9] (2011) Disco mapreduce framework. [Online]. Available: http://discoproject.org

[10] (2011) Hadoop - facebook engg, note. [Online]. Available: http://www.facebook.com/note.php?noteid=16121578919

[11] (2011) Yahoo inc. hadoop at yahoo! [Online]. Available: http://developer.yahoo.com/hadoop

[12] T. Gunarathne, T. Wu, J. Qiu, and G. Fox, "Cloud computing paradigms for pleasingly parallel biomedical applications," in *Proceedings of 19th ACM International Symposium on High Performance Distributed Computing*. ACM, January 2010, pp. 460–469.

[13] S. Krishnan, C. Baru, and C. Crosby, "Evaluation of mapreduce for gridding lidar data," in *Proceedings of the CLOUDCOM '10*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 33–40.

[14] G. Venter and J. Sobieszczanski-Sobieski, "A parallel particle swarm optimization algorithm accelerated by asynchronous evaluations," *Journal of Aerospace Computing, Information, and Communication*, 2005.

[15] M. Ismail, "Parallel genetic algorithms (pgas): master slave paradigm approach using mpi," in *E-Tech 2004*, july 2004, pp. 83 – 87.

[16] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing*. Addison-Wesley, USA, 2003.

[17] A. McNabb, C. Monson, and K. Seppi, "Parallel pso using mapreduce," in *IEEE Congress on Evolutionary Computation*, Sept. 2007, pp. 7–14.

[18] C. Jin, C. Vecchiola, and R. Buyya, "Mrpga: An extension of mapreduce for parallelizing genetic algorithms," in *Proceedings of the 2008 Fourth IEEE International Conference on eScience*, ser. ESCIENCE '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 214–221.

[19] B. Wu, G. Wu, and M. Yang, "A mapreduce based ant colony optimization approach to combinatorial optimization problems," in *Natural Computation (ICNC), 2012 Eighth International Conference on*, may 2012, pp. 728 –732.

[20] K. Krishnanand and D. Ghose, "Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions," *Swarm Intelligence*, vol. 3, pp. 87–124, 2009.