Artificial Bee Colony Algorithm on Distributed Environments

Anan Banharnsakun, Tiranee Achalakul, Booncharoen Sirinaovakul
Department of Computer Engineering
King Mongkut's University of Technology Thonburi
Bangkok, Thailand
smilecolon@yahoo.com, tiranee@cpe.kmutt.ac.th, boon@kmutt.ac.th

Abstract— **Artificial Bee Colony (ABC) is a metaheuristic approach in which a colony of artificial bees cooperates in finding good solutions for numerical optimization problems. ABC is adopted widely for use in several domains of solution optimization. However, the algorithm generally requires a considerably large computational time and resources. In order to enhance the performance of this algorithm for a large problem size, we introduce a distributed version of ABC. In our parallel algorithm, the entire bee colony is decomposed into several subgroups. Each subgroup then performs a local search concurrently on each processor node. The local best solutions are then exchanged among processor nodes. The algorithm implementation utilizes the message passing technique as a communication paradigm. We then empirically assess the performance based on both result accuracy and algorithm's efficiency. The experimental results show improvement in both solution quality and computing time when comparing to the sequential ABC algorithm.**

Keywords- Artificial Bee Colony (ABC); Swarm Intelligence; Numerical Optimization; Distributed Environments; Parallel Computing.

I. INTRODUCTION

Many mathematical methods are designed to solve the optimization problem class. However, in many real-world problems, there exist a large computational and resource demands and a single processor may not be computationally sufficient. An optimization approach based on a parallel implementation is thus an alternative way to obtain the enhanced computational throughput and the global search capability.

Parallel implementation can roughly be categorized into two models [1]: shared memory and distributed memory. Although a shared memory model provides a global address space that makes parallel programming easier, the lack of system scalability can hinder a large scale development. In such cases, a distributed memory model is usually considered despite the programming difficulty related to memory management and data communication.

Metaheuristic approaches, such as Genetic Algorithm (GA), Ant Colony Optimization (ACO), and Particle Swarm Optimization (PSO), are popular optimization methods that have been implemented on a parallel system. Many researchers have developed numerous parallel algorithms in order to increase the efficiency. To produce a faster GA-based shortest part routing algorithm, a coarse-grained Parallel Genetic Algorithm was proposed by Yussof, et al. [2]. In this work, all computing nodes randomly created their own sub-population and each executed sequential GA independently. The migration scheme was then employed to increase the sub-population diversity by exchanging solutions between computing nodes.

The parallel Ant Colony Optimization algorithm was presented by Jie, et al. [3]. In this work, the computational time was reduced and the significant speedup was obtained through coarse-granularity and partially asynchronous parallel technique. Based on this method, ant colony was divided into sub-colonies and each sub-colony was deployed on a computing node. Master node gathered the local best solution from all slave nodes and identified the node id with the best solution. Finally, the best pheromone matrix and solution were broadcasted to the rest of the participating nodes.

Li and Wada [4] introduced a parallel version of the Particle Swarm Optimization algorithm for solving complex large-scale optimization problems. A communication method called delayed exchange was presented in the work. In this approach, each processor used its current local best solution and local best solutions computed previously by the other computing nodes to calculate the global best solution.

The Artificial Bee Colony (ABC) algorithm introduced by D. Karaboga [5] was one of the metaheuristic approaches that has been used to find an optimal solution in several optimization problems [6,7,8]. This algorithm is inspired by the behavior of honey bees when seeking a quality food source. It is simple in concept and easy to implement because it only uses one parameter called a "limit" [9] for controlling the abandoned solution in its algorithm.

After an extensive literature survey, we found only one literature proposed by H. Narasimhan [10] that focused on a parallel implementation of the ABC algorithm. However, the proposed method is based on a shared memory concept which posted a limitation on the hardware scalability. In order to improve the scalability, we designed a parallel ABC algorithm for distributed memory systems. The quality of the computed solutions and the efficiency of proposed algorithm are addressed in this paper.

This paper is organized as follows. Section II presents a brief overview of the ABC Algorithm. Section III proposes the design of ABC algorithm on distributed environments. Section IV outlines parameters setting of experiments, summarizes the results and discusses the performance of the proposed algorithm. Section V draws a conclusion.

978-1-4244-7376-2/10/$26.00

II. THE ABC ALGORITHM

For more understanding on our distributed ABC algorithm, this section provides the knowledge background of the ABC algorithm and its sequential method for solving the optimization problems.

The ABC algorithm assumes the existence of a set of computational agents called honey bees. The honey bees in this algorithm are categorized into three groups: employed bees, onlooker bees and scout bees. The colony is equally separated into employed bees and onlooker bees. Each solution in the search space consists of a set of optimization parameters which represent a food source position. The number of employed bees is equal to the number of food sources. In other words, there is only one employed bee on each food source. The quality of food source is called its "fitness value" and is associated with its position. The process of bees seeking for good food sources is the process used to find the optimal solution. The sequential method of the ABC algorithm can be shown as Fig. 1 below.

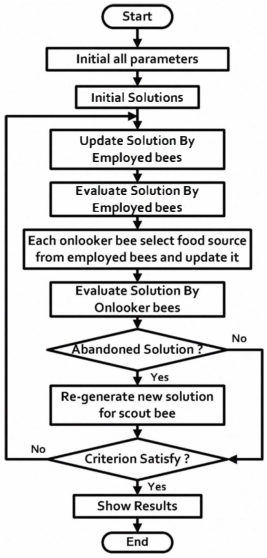

Figure 1. The sequential method of the ABC algorithm.

From Fig. 1, the employed bees will be responsible for investigating their food sources and sharing the information about these food sources to recruit the onlooker bees. The onlooker bees will make a decision to choose a food source based on this information. The food source that has higher quality will have a larger chance to be selected by onlooker bees than one of lower quality. An employed bee whose food source is rejected as low quality by employed and onlooker bees will change to a scout bee to search randomly for new food sources. By this mechanism, the exploitation will be handled by employed and onlooker bees while the exploration will be maintained by scout bee. The details of the algorithm are as follows.

First, randomly distributed initial food source positions are generated. The process can be represented by (1) below.

$$F(x_i), x_i \in R^D, i \in \{1,2,3..,SN\}, \quad (1)$$

x_i is a position of food source as a D-dimensional vector, $F(x_i)$ is the objective function which determines how good a solution is, and SN is the number of food sources.

After initialization, the population is subjected to repeated cycles of three major steps: updating feasible solutions, selecting feasible solutions, and avoiding suboptimal solutions.

In order to update feasible solutions, all employed bees select a new candidate food source position. The choice is based on the neighborhood of the previously selected food source. The position of the new food source is calculated from (2) below.

$$v_{ij} = x_{ij} + \emptyset_{ij}(x_{ij} - x_{kj}) \quad (2)$$

v_{ij} is a new feasible solution that is modified from its previous solution value (x_{ij}) based on a comparison with the randomly selected position from its neighboring solution (x_{kj}). $\emptyset_{ij}$ is a random number between [-1,1] which is used to randomly adjust the old solution to become a new solution in the next iteration. $k \in \{1,2,3..,SN\}$ and $j \in \{1,2,3..,D\}$ are randomly chosen indexes. The difference between x_{ij} and x_{kj} is a difference of position in a particular dimension. The algorithm changes each position in only one dimension in each iteration. Using this approach, the diversity of solutions in the search space will increase in each iteration.

The old food source position in the employed bee's memory will be replaced by the new candidate food source position if the new position has a better fitness value. Employed bees will return to their hive and share the fitness value of their new food sources with the onlooker bees.

In the next step, each onlooker bee selects one of the proposed food sources depending on the fitness value obtained from the employed bees. The probability that a food source will be selected can be obtained from (3) below.

$$P_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_n} \quad (3)$$

Where fit_i is the fitness value of the food source i, which is related to the objective function value ($F(x_i)$) of the food source i.

The probability of a food source being selected by the onlooker bees increases as the fitness value of a food source increases. After the food source is selected, onlooker bees will go to the selected food source and select a new candidate food source position in the neighborhood of the selected food source. The new candidate food source can be expressed and calculated by (2).

In the third step, any food source position that does not improve the fitness value will be abandoned and replaced by a new position that is randomly determined by a scout bee. This helps avoid suboptimal solutions. The new random position chosen by the scout bee will be calculated from (4) below.

$$x_{ij} = x_j^{min} + rand[0,1] * \left(x_j^{max} - x_j^{min}\right) \quad (4)$$

Where x_j^{min} is the lower bound of the food source position in dimension j and x_j^{max} is the upper bound of the food source position in dimension j.

The maximum number of cycles (MCN) is used to control the number of iterations and is a termination criterion. The process will be repeated until the output of the objective function reaches a defined threshold value or the number of iteration equals the MCN. A defined threshold value is set to be equal to the global minimum value or the global maximum value depending on the type of the optimization problems.

III. THE DESIGN OF ABC ON DISTRIBUTED ENVIRONMENTS

The foraging behavior of bees is a distributed process in nature. Each forage bee seeks food sources concurrently around their hive. The collective decision is gradually made during the selection of food sources. This foraging behavior can thus be modeled straightforwardly for distributed environments.

The bees independently evaluate the quality of different new candidate food sources on their own by using information advertised by other bees. This concept is mapped into our proposed method. We divide bees into subgroups. Each subgroup then seeks a quality food source and exchanges this information with other subgroups.

In our distributed ABC, the Manager-worker model [1] is utilized for process management. The model consists of one manager and multiple workers. The manager is usually responsible for initialization, load balancing, decomposing the bee colony, distributing the subgroups of bees to workers, and collecting and displaying the global results. The workers, on the other hand, process the main computations. They are responsible for receiving the subgroups of bees from the manager, process them, and return the local best solution results in the last iteration. The inter-processor communications in our model occur between the manager and the workers and among the workers. The communications are implemented based on the message-passing interface (MPI), which is a standard specification for the message-passing functions. MPI provides a collection of routines which are embedded in application codes including sending, receiving, and other message passing operations. MPI uses objects called communicators or a process group to define the collection of processes for computation (manager and workers).

Within the communicator, each process is ranked by a unique integer between 0 and a number of processes – 1. These ranks are used as process identifiers. In our work, rank 0 is given to the manager. A complete distributed ABC code is loaded on each process which can be illustrated in the Fig. 2 below.

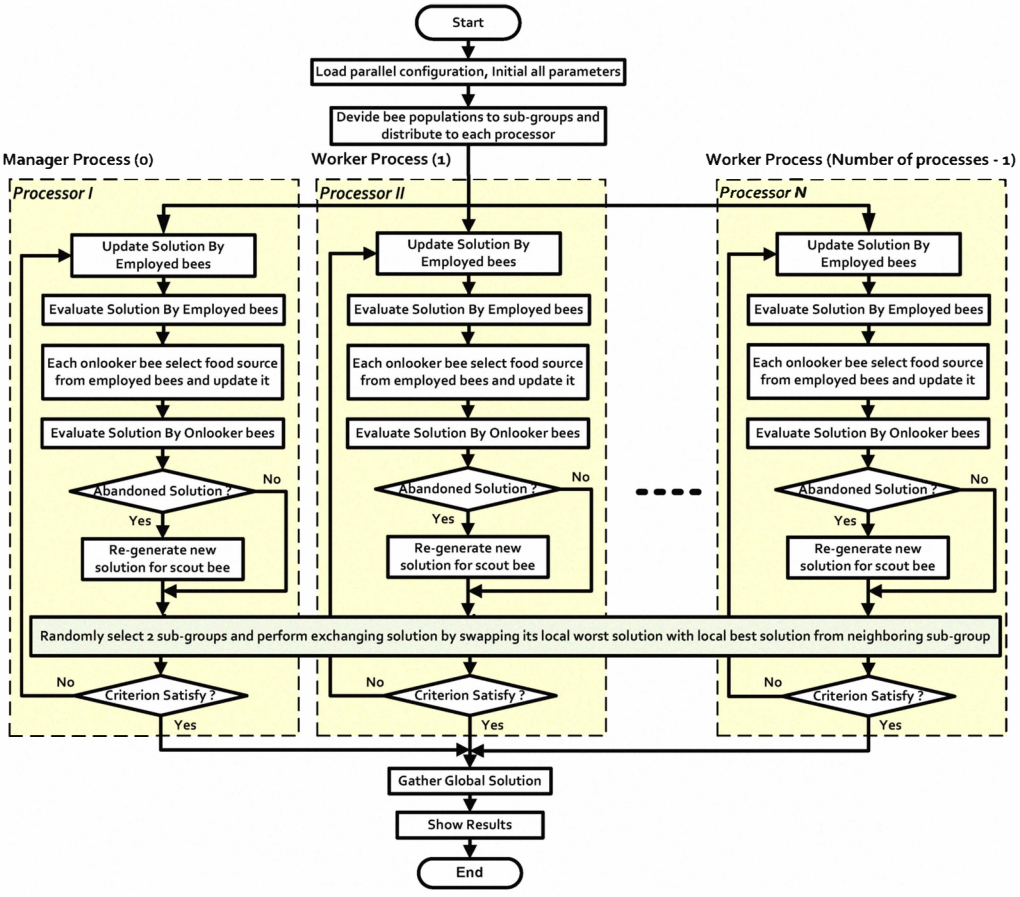

Figure 2. The Distributed ABC Flowchart.

The manager process decomposes the entire colony of bees into a set of subgroups. It then assigns these subgroups to each worker process. While not coordinating the computation, the manager also performs the useful work the same way the workers do. In each iteration, the set of bees in each subgroup concurrently improves the solutions.

In the last step of each iteration, the manager process randomly picks two subgroups from the entire population. These two subgroups then exchange their local best solutions. The local best solution from one subgroup will replace the local worst solution on another subgroup. This process can be shown in Fig. 2. The algorithm only exchanges information at this synchronization point. The communication overhead is thus small.

In the last iteration, the manager process gathers the local best solutions from all worker processes and calculates the global best solution.

While the sequential method uses only one random seed at the initial step, the distributed method employs a number of random seeds equal to the number of subgroups. This mechanism will increase the diversity of candidate solutions. In other words, the exploration process on search space will be improved and chance to find the optimal solution is thus higher than the sequential method. Moreover, the solution can converge more quickly.

IV. Experiment Setting and Results

Our experiments aim to evaluate the performance of the proposed parallel algorithm in comparison to the sequential implementation of ABC in both the perspectives of result accuracy and algorithm's efficiency. We implemented the algorithm in C++ and executed it on a 4-machine distributed environment. Each machine has the processing speed of 2.8 GHz and 4 GB memory.

A. The Evaluation of Solution Accuracy

To evaluate the solution accuracy, the distributed ABC algorithm was tested on six benchmark functions. The parameter ranges on each function can be shown in Table I.

In this experiment, the effect of the periodic time to perform the local best solution exchanging process ($P_{exchange}$) was also observed. In order to investigate this effect on the output quality, we divided the experiments into two sets. In the first set, this exchanging process was performed every 100 iterations ($P_{exchange}$ = 100). Next, it was performed every 20 iterations ($P_{exchange}$ = 20) in the second set.

In both experiment sets, the number of employed and onlooker bees were set to 125, the dimension on each benchmark function was set to 30 except for the Schaffer function, where the dimension is set to 2. The MCN was set to 2000. Each of the experiments was repeated 100 times with different random seeds. The objective of the search process is to find the solutions that can produce the minimum output value from these benchmark functions. In other words, the aim is to minimize $f_i(\vec{x})$ in Table I.

The mean and the standard deviation of the output values were then recorded. The results obtained for the first and the second experiment sets are shown in Table II and III respectively. Note that lower values in both "Mean" and "SD" columns indicate better solutions.

The comparisons of the convergence speed in term of number of iterations obtained from the first experiment set are shown in Fig. 3.

From the results, we can see that the mean and the standard deviation of the output values obtained from our distributed ABC were in the same range as those obtained from sequential ABC on all benchmark functions. The number of iterations required to reach the convergent point are also similar between the distributed and sequential versions. Moreover, when the periodic time to exchange the local best solution was increased, the solution quality was improved. The average improvement on the solution quality was 56% when compared to the sequential version.

TABLE I. Numerical Benchmark Functions

Function Name	Function	Characteristic	Ranges	Global Minimum
Sphere	$f_1(\vec{x}) = \sum_{i=1}^{D} x_i^2$	Uni-modal, Separable	$-100 \le x_i \le 100$	0
Griewank	$f_2(\vec{x}) = \frac{1}{4000}\left(\sum_{i=1}^{D}(x_i^2)\right) - \left(\prod_{i=1}^{D}\cos\left(\frac{x_i}{\sqrt{i}}\right)\right) + 1$	Multi-modal, Non-Separable	$-600 \le x_i \le 600$	0
Rastrigin	$f_3(\vec{x}) = \sum_{i=1}^{D}(x_i^2 - 10\cos(2\pi x_i) + 10)$	Multi-modal, Separable	$-5.12 \le x_i \le 5.12$	0
Rosenbrock	$f_4(\vec{x}) = \sum_{i=1}^{D} 100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2$	Uni-modal, Non-Separable	$-30 \le x_i \le 30$	0
Ackley	$f_5(\vec{x}) = 20 + e - 20e^{(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2})} - e^{\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)}$	Multi-modal, Non-Separable	$-30 \le x_i \le 30$	0
Schaffer	$f_6(\vec{x}) = 0.5 + \frac{(\sin\sqrt{x_1^2 + x_2^2})^2 - 0.5}{(1 + 0.001(x_1^2 + x_2^2))^2}$	Multi-modal, Non-Separable	$-100 \le x_i \le 100$	0

TABLE II. RESULT OBTAINED FROM THE FIRST EXPERIMENT SET ($P_{EXCHANGE}$ = 100)

Function Name	ABC		Distributed ABC					
	#Processor = 1		#Processors = 2		#Processors = 3		#Processors = 4	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
Sphere	3.64E-26	3.77E-26	8.20E-27	6.31E-27	9.93E-27	7.59E-27	1.13E-26	8.66E-27
Griewank	7.44E-22	2.41E-21	5.67E-23	7.01E-23	1.05E-22	1.17E-22	1.51E-22	2.77E-22
Rastrigin	2.50E-22	1.50E-22	3.45E-23	2.81E-23	9.51E-23	7.81E-23	1.08E-22	9.33E-23
Rosenbrock	3.05E-02	7.06E-02	3.98E-02	5.09E-02	3.95E-02	4.34E-02	3.80E-02	4.61E-02
Ackley	1.55E-13	2.55E-14	1.10E-13	1.69E-14	1.29E-13	2.25E-14	1.32E-13	2.91E-14
Schaffer	7.29E-12	4.20E-11	1.12E-13	1.05E-12	4.85E-14	4.82E-13	7.65E-14	7.51E-13

TABLE III. RESULT OBTAINED FROM THE SECOND EXPERIMENT SET ($P_{EXCHANGE}$ = 20)

Function Name	ABC		Distributed ABC					
	#Processor = 1		#Processors = 2		#Processors = 3		#Processors = 4	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
Sphere	3.64E-26	3.77E-26	1.00E-28	1.02E-28	4.75E-29	5.00E-29	4.22E-29	3.11E-29
Griewank	7.44E-22	2.41E-21	3.00E-27	2.72E-26	1.42E-26	4.43E-26	2.00E-26	7.50E-26
Rastrigin	2.50E-22	1.50E-22	1.19E-27	2.19E-27	1.23E-26	1.58E-26	1.37E-26	1.43E-26
Rosenbrock	3.05E-02	7.06E-02	5.72E-02	7.89E-02	7.01E-02	9.46E-02	7.15E-02	8.16E-02
Ackley	1.55E-13	2.55E-14	7.58E-14	9.91E-15	5.86E-14	8.24E-15	6.37E-14	8.18E-15
Schaffer	7.29E-12	4.20E-11	3.98E-50	4.10E-51	7.07E-50	7.10E-49	7.17E-47	7.20E-48

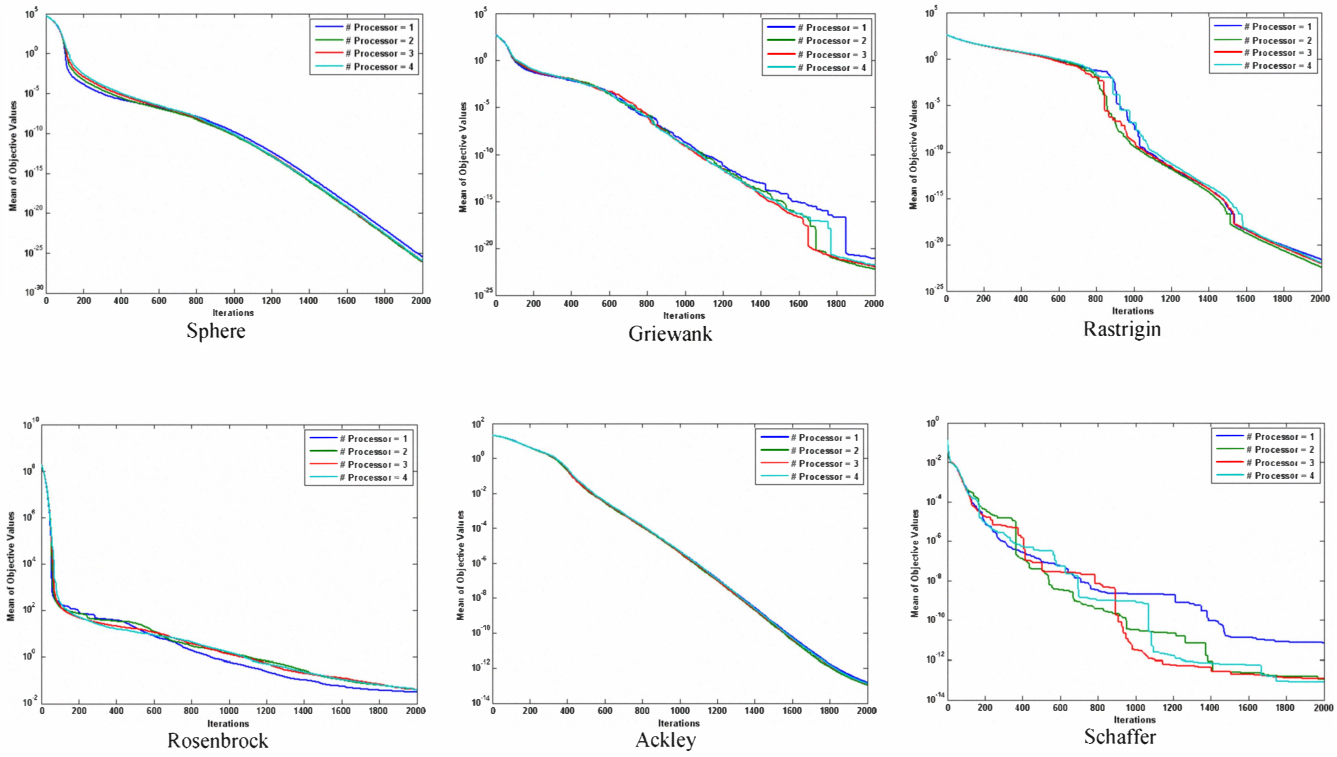

Figure 3. Number of Iterations to convergence of six benchmark functions.

B. *The Evaluation of Algorithm's Efficiency*

In this section, the proposed algorithm's efficiency was evaluated on a large problem size and the parallel scalability analysis is discussed.

To ensure that the work load in a single iteration is substantially high, we chose Griewank as a benchmark function. The number of employed and onlooker bees were set to 1000, the dimension on this function was set to 1000, and the MCN was set to 10000.

The most common measurements of parallel algorithm effectiveness are speedup and efficiency. These values can be calculated from (5) and (6) respectively.

$$Speedup = \frac{\text{Execution time on one processor core}}{\text{Execution time on } m \text{ processor cores}} \quad (5)$$

$$Efficiency = \frac{Speedup}{m} \quad (6)$$

The results of the execution time, speedup and efficiency for different number of processors are shown in Table IV and the scalability plot of these results are illustrated in Fig. 4 below.

TABLE IV. SPEEDUP AND EFFICIENCY FOR GIVEN NUMBER OF PROCESSORS

Number of processors	Execution Time (Sec)	Speedup	Efficiency
1	3547.407	-	-
2	1776.578	1.997	0.998
3	1192.031	2.976	0.992
4	915.266	3.876	0.969

With a large problem size, our distributed ABC algorithm performed very close to ideal time. The execution times dropped almost linearly as more processors were added to the computation, while the solution quality was maintained. The computation to communication ratio was also high. In other words, low communication overhead was observed. The efficiency could thus be measured at as high as 99% in most cases. It can be concluded that by adding more processors, a very large problem size can be solved in a relatively short amount of time using a cluster of state of the art machines.

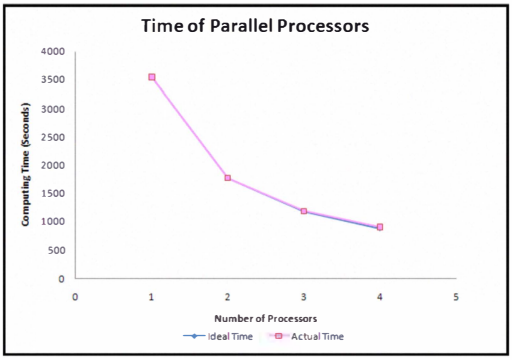

Figure 4. Scalability plot.

V. CONCLUSIONS

In this paper, a parallel ABC algorithm for distributed environments was proposed. The algorithm was designed based on the manager-worker model. Process synchronization and communication are implemented based on the message-passing interface (MPI). The mechanism to exchange the local best solution among computing nodes was introduced. The proposed algorithm was evaluated on both the perspectives of result accuracy or algorithm's efficiency. The numerical results indicate that the distributed ABC algorithm can produce similar solutions as sequential ABC. Moreover, with a large problem size, our parallel method achieves a near linear speedup with the efficiency close to 100%. The optimization problem can thus be solved faster, which will improve the practicality of ABC.

ACKNOWLEDGMENT

This work is supported by the Thailand Research Fund through the Royal Golden Jubilee Ph.D. Program under Grant No. PHD/0038/2552.

REFERENCES

[1] A. Grama, A. Gupta, G. Karypis, V. Kumar, *Introduction to parallel computing*. New York: Addison-Wesley, 2003.

[2] S. Yussof, R.A. Razali, O.H. See, A.A. Ghapar, M.M. Din, "A Coarse-Grained Parallel Genetic Algorithm with Migration for Shortest Path Routing Problem," *IEEE International Conference on High Performance Computing and Communications*, 2009, pp. 615-621.

[3] X. Jie, L. CaiYun, C. Zhong, "A New Parallel Ant Colony Optimization Algorithm Based on Message Passing Interface," *Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, 2008, pp. 178-182.

[4] B. Li, K. Wada, "Parallelizing particle swarm optimization," *IEEE Pacific Rim Conference on Communications, Computers and signal Processing*, 2005, pp. 288-291.

[5] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Erciyes University, Engineering Faculty, Computer Engineering Department, Turkey, Technical Report-TR06, 2005.

[6] N. Karaboga, "A new design method based on artificial bee colony algorithm for digital IIR filters," *Journal of the Franklin Institute.*, vol. 346, pp. 328-348, 2009.

[7] F. Kang, J. Li, Q. Xu, "Structural inverse analysis by hybrid simplex artificial bee colony algorithms," *Computers and Structures.*, vol. 87, pp. 861-870, 2009.

[8] A. Singh, "An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem," *Applied Soft Computing.*, vol. 9, pp. 625-631, 2009.

[9] D. Karaboga, B. Basturk, "On the performance of artificial bee colony (ABC) algorithm," *Applied Soft Computing.*, vol. 8, pp. 687–697, 2008.

[10] H. Narasimhan, "Parallel artificial bee colony (PABC) algorithm," *World Congress on Nature & Biologically Inspired Computing*, 2009, pp. 306-311.