Particle Swarm Approach to Scheduling Work-Flow Applications in Distributed Data-Intensive Computing Environments

Hongbo Liu[1,2], Shichang Sun[2], Ajith Abraham[1,3]
[1]School of Computer Science, Dalian Maritime University, Dalian, 116024, China
[2]Department of Computer Science, Dalian University of Technology, Dalian, 116023, China
[3]School of Computer Science and Engineering, Chung-Ang University, Seoul 156-756, Korea
lhb@dlut.edu.cn, schsunster@gmail.com, ajith.abraham@ieee.org

Abstract

The scheduling problem in distributed data-intensive computing environments has been an active research topic due to immense practical applications. In this paper, we model the scheduling problem for work-Flow applications in distributed Data-intensive computing environments (FDSP) and make an attempt to formulate and solve the problem using a particle swarm optimization approach. We illustrate the algorithm performance and trace its feasibility and effectiveness with the help of an example.

1. Introduction

With the development of the High Performance Computing (HPC), Grid, etc., some complex applications are designed by communities of researchers in domains such as high-energy physics, astronomy, biology [1, 2, 3, 4] and Human Brain Planning (HBP) [5]. For implementing and utilizing successfully these applications, one of the most important activity is to find appropriate schedules before the application is executed. The goal is to find an optimal assignment of tasks in the applications with respect to the costs of the available resources. However, the scheduling problem in distributed data-intensive computing environments seems quite different from the one in the traditional situation. Scheduling jobs and resources in these data-intensive applications need to meet the specific requirements, including process flow, data access/transfer, completion cost, flexibility and availability. All kinds of components in the application can interact with each other directly or indirectly. Scheduling algorithms using traditional computing paradigms barely consider the data transfer problem during mapping computational tasks, and this neglect might be costly in distributed data-intensive applications [6].

In this paper, we explore the scheduling problem for work-flow applications in distributed data-intensive computing environments. This paper is organized as follows. We model and formulate the problem in Section 2. The proposed approach based on particle swarm algorithm is presented in Section 3. In Section 4, experiment results and discussions are provided in detail. Finally, we conclude our work in the paper.

2. Problem formulation

The scheduling problem in distributed data-intensive computing environments has been an active research topic, and therefore many terminologies have been suggested. Unfortunately, some of the terms are neither clearly stated nor consistently used by different researchers, which frequently makes readers confused. For clarity purposes, some key terminologies are defined as follows:

- Machine (computing unit)
  Machine (computing unit) is a set of computational resources with limited capacities. It may be a simple personal machine, a workstation, a super-computer, or a cluster of workstations. The computational capacity of the machine depends on its number of CPUs, amount of memory, basic storage space and other specializations. In other words, each machine has its calculating speed, which can be expressed in number of Cycles Per Unit Time (CPUT).

- Data Resource
  Data resources are the datasets which effect the scheduling. They are commonly located on various storage repositories or data hosts. Data resources are connected to the computational resources (machines) by links of different bandwidths.

- Job and Operation
  A job is considered as a single set of multiple atomic

Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications (ISDA'06)
0-7695-2528-8/06 $20.00 © 2006 IEEE

operations/tasks. Each operation will be typically allocated to execute on one single machine without preemption. It has input and output data, and processing requirements in order to complete its task. One of the most important processing requirements is the work-flow, which is the ordering of a set of operations for a specific application. These operations can be started only after the completion of the previous operations from this sequence, which is the so-called work-flow constraints. The operation has the processing length in number of cycles.

- Work-Flow Application
  A work-flow application consists of a collection of interacting components that need to be executed in a certain partial order for solving successful a certain problem. The components involve a number of dependent or independent jobs, machines, the bandwidth of the network, etc. They have specific control and data dependency between them.

- Schedule and Scheduling Problem
  A schedule is the mapping of the tasks to specific time intervals of machines. A scheduling problem is specified by a set of machines, a set of jobs/operations, optimality criteria, environmental specifications, and by other constraints. The scheduling problem for work-flow applications in distributed Data-intensive computing environments is abbreviated to "FDSP".

To formulate the scheduling problem, suppose a work-flow application comprises of q Jobs $\{J_1, J_2, \cdots, J_q\}$, m Machines $\{M_1, M_2, \cdots, M_m\}$ and k Data hosts $\{D_1, D_2, \cdots, D_k\}$. In the application, the calculating speed of the machine are $\{P_1, P_2, \cdots, P_m\}$. Each job consists of a set of operations $J_j = \{O_{j,1}, O_{j,2}, \cdots, O_{j,p}\}$. For convenience, we will decompose all the jobs to atomic operations and re-sort the operations as $\{O_1, O_2, \cdots, O_n\}$. Their processing lengths are $L_1, L_2, \cdots, L_n$, respectively. All the operations are in the specific work-flow, and they will be carried orderly out on the machines with data retrieval, data input and data output. The operations in the work-flow can be represented as or be transformed to a Directed Acyclic Graph (DAG), where each node in the DAG represents an operation and the edges denote control/data dependencies.

Definition 1 *A work-flow graph for for data-intensive work-flow applications can be represented as $G = (O, E)$, where the set of nodes $O = \{O_1, O_2, \cdots, O_n\}$ corresponds to the set of operations to be executed, and the set of weighted, directed edges E represents both the precedence constraints and the data transfers volume among operations in O. An edge $(O_i, O_j) \in E$ implies that O_j can not start execution until O_i finishes and sends its result to O_j. We call task O_i predecessor of task O_j and task O_j successor of task O_i. Let $Pred(O_i)$ denote the set of all the predecessors of task O_i. Let $Succ(O_i)$ denote the set of all the successors of task O_i.*

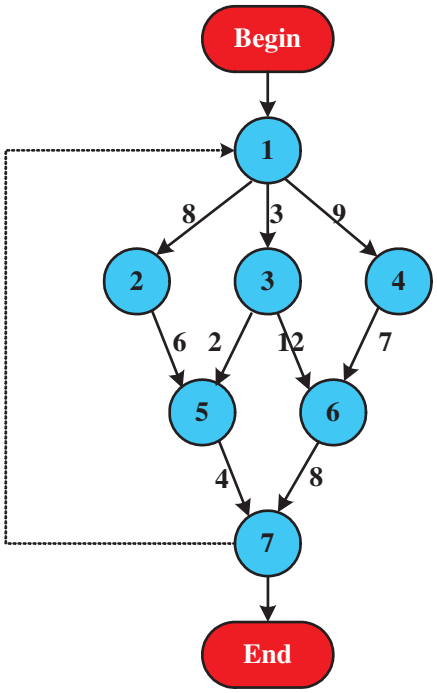

Figure 1. An instance with 7 operations.

The relation between the operations can be represented by a flow matrix $F = [f_{i,j}]$, in which the element $f_{i,j}$ stores the weight value if the edge $< O_i, O_j >$ is in the graph, otherwise it is set to "-1". Figure 1 depicts a work-flow instance of 7 operations. The recursive loop between O_1 and O_7 can be neglected when the scheduling focused on the stage within the loop. Its flow matrix F is

$$\begin{bmatrix} -1 & 8 & 3 & 9 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & 6 & -1 & -1 \\ -1 & -1 & -1 & -1 & 2 & 12 & -1 \\ -1 & -1 & -1 & -1 & -1 & 7 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & 4 \\ -1 & -1 & -1 & -1 & -1 & -1 & 8 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

The data host dependencies of the operations are determined by the retrieval matrix $R = [r_{i,j}]$. The element $r_{i,j}$ is the retrieval time which O_i executes retrieval processing on the data host D_j. There are the other two matrices $A = [a_{i,j}]$ and $B = [b_{i,j}]$, where the element $a_{i,j}$ in the former is the distance between the machine M_i and M_j, and the element $b_{i,j}$ in the latter is the distance between the

machine M_i and the data host D_j. For each operation, its completion time is the sum of three components: the input data time, the retrieval data time, and the execution time on the assigned machine. It is to be noted that the input data time can be started to accumulate only after the completion of the previous operations in the work-flow. Given a feasible solution $S = \{S_1, S_2, \cdots, S_n\}$, S_i is the serial number of the machine which the operation O_i is assigned on. Define C_{O_i} ($i \in \{1, 2, \cdots, n\}$) as the completion time that the machine M_{S_i} completes the operation O_i. For the operation O_i, its completion time C_{O_i} can be calculated by Eq. (1).

$$C_{O_i} = \sum_{\substack{l=1 \\ f_{l,i} \neq -1}}^{n} f_{l,i} a_{S_l,S_i} \sum_{h=1}^{k} r_{i,h} b_{S_i,h} + L_i / P_{S_i} \quad (1)$$

To formulate the objective, $\sum C_{M_i}$ represents the time that the machine M_i completes the processing of all the operations assigned on it. Define $C_{max} = max\{\sum C_{M_i}\}$ as the makespan, and $C_{sum} = \sum_{i=1}^{m}(\sum C_{M_i})$ as the flowtime. The scheduling problem is thus to both determine an assignment and a sequence of the operations on all machines that minimize some criteria. Most important optimality criteria are to be minimized:

1. the maximum completion time (makespan): C_{max};
2. the sum of the completion times (flowtime): C_{sum}.

Minimizing C_{sum} asks the average operation is finished quickly, at the expense of the largest operation taking a long time, whereas minimizing C_{max} asks that no operation takes too long, at the expense of most operations taking a long time. Minimization of C_{max} would result in maximization of C_{sum}. The weighted aggregation is the most common approach to the problems. According to this approach, the objectives, $F_1 = min\{C_{max}\}$ and $F_2 = min\{C_{sum}\}$, are summed to a weighted combination:

$$F = w_1 min\{F_1\} + w_2 min\{F_2\} \quad (2)$$

where w_1 and w_2 are non-negative weights, and $w_1 + w_2 = 1$. These weights can be either fixed or adapt dynamically during the optimization. The fixed weights, $w_1 = w_2 = 0.5$, are used in this paper. Alternatively, the weights can be changed gradually according to the Eqs. (3) and (4). The alternate curves ($R = 200$) are showed in Figure 2. In fact, the dynamic weighted aggregation [7] mainly takes F_1 into account. Because F_2 is commonly much larger than F_1, the final solution would have a large weight ($w_1 \rightarrow 1$) on F_1 during minimizing the objective.

$$w_1(t) = |sin(2\pi t/R)| \quad (3)$$

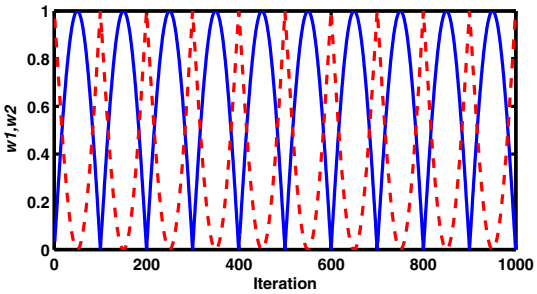

Figure 2. The dynamic weight curves.

$$w_2(t) = 1 - w_1(t) \quad (4)$$

Definition 2 *A scheduling problem for data-intensive work-flow applications can be defined as* $\prod = (J(L), M(P), D, G, R, A, B, f)$. *If all the jobs are decomposed to atomic operations and the relationship between the operations are transformed to the flow matrix* F, *the scheduling problem can be represented as* $\prod = (O(L), M(P), D, F, R, A, B, f)$. *The key components are operations, machines and data-hosts. For the sake of simplify, the scheduling problem also be represented in triple* $T = (O, N, D)$.

3. Particle swarm heuristic for FDSP

Particle swarm algorithm is inspired by social behavior patterns of organisms that live and interact within large groups. In particular, it incorporates swarming behaviors observed in flocks of birds, schools of fish, or swarms of bees, and even human social behavior, from which the Swarm Intelligence (SI) paradigm has emerged [8, 9]. It could be implemented and applied easily to solve various function optimization problems, or the problems that can be transformed to function optimization problems. As an algorithm, its main strength is its fast convergence, which compares favorably with many other global optimization algorithms [10, 11, 12]. The classical particle swarm model consists of a swarm of particles, which are initialized with a population of random candidate solutions. They move iteratively through the d-dimension problem space to search the new solutions, where the fitness f can be calculated as the certain qualities measure. Each particle has a position represented by a position-vector $\vec{x}_i$ (i is the index of the particle), and a velocity represented by a velocity-vector $\vec{v}_i$. Each particle remembers its own best position so far in a vector $\vec{x}_i^{\#}$, and its j-th dimensional value is $x_{ij}^{\#}$. The best position-vector among the swarm so far is then stored in a vector $\vec{x}^*$, and its j-th dimensional value is x_j^*. During the iteration time t, the update of the velocity from the previous

Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications (ISDA'06)
0-7695-2528-8/06 $20.00 © 2006 IEEE

velocity to the new velocity is determined by Eq.(5). The new position is then determined by the sum of the previous position and the new velocity by Eq.(6).

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_1(x_{ij}^{\#}(t) - x_{ij}(t)) + c_2 r_2(x_j^*(t) - x_{ij}(t)) \quad (5)$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \quad (6)$$

In the particle swarm model, the particle searches the solutions in the problem space within a range $[-s, s]$ (If the range is not symmetrical, it can be translated to the corresponding symmetrical range.) In order to guide the particles effectively in the search space, the maximum moving distance during one iteration is clamped in between the maximum velocity $[-v_{max}, v_{max}]$ given in Eq.(7), and similarly for its moving range given in Eq.(8):

$$v_{i,j} = sign(v_{i,j}) min(|v_{i,j}|, v_{max}) \quad (7)$$

$$x_{i,j} = sign(x_{i,j}) min(|x_{i,j}|, x_{max}) \quad (8)$$

The value of v_{max} is $\rho \times s$, with $0.1 \leq \rho \leq 1.0$ and is usually chosen to be s, i.e. $\rho = 1$. The pseudo-code for particle swarm optimization algorithm is illustrated in Algorithm 1.

Algorithm 1 Particle Swarm Algorithm

01. Initialize the size of the particle swarm n, and other
02. parameters; Initialize the positions and the velocities
03. for all the particles randomly.
04. While (the end criterion is not met) do
05. $t = t + 1$;
06. Calculate the fitness value of each particle;
07. $\vec{x}^* = argmin_{i=1}^{n}(f(\vec{x}^*(t-1)), f(\vec{x}_1(t)),$
08. $f(\vec{x}_2(t)), \cdots, f(\vec{x}_i(t)), \cdots, f(\vec{x}_n(t)));$
09. For i= 1 to n
10. $\vec{x}_i^{\#}(t) = argmin_{i=1}^{n}(f(\vec{x}_i^{\#}(t-1)), f(\vec{x}_i(t)));$
11. For j = 1 to d
12. Update the j-th dimension value of $\vec{x}_i$ and $\vec{v}_i$
13. according to Eqs.(5),(7),(6),(8);
14. Next j
15. Next i
16. End While.

For applying particle swarm algorithm successfully for the FDSP problem, one of the key issues is how to map the problem solution to the particle space, which directly affects its feasibility and performance [13, 14]. We setup a search space of n dimension for an $(n - Operations, m - Machines)$ FDSP problem. Each dimension is limited to $[1, m+1)$. For example, consider the $(7 - Operations, 3 - Machines)$ FDSP, Figure 3 illustrates how to map one possible assignment to one particle position coordinates in the particle swarm domain. Each dimension of the particle's position maps one operation, and the value of the position indicates the machine number to which this task/operation is assigned during the course of particle swarm algorithm. So the value of a particle's position should be integer. But after updating the velocity and position of the particles, the particle's position may appear real values such as 1.4, etc. It is meaningless for the assignment. Therefore, in the algorithm we usually round off the real optimum value to its nearest integer number. By this way, we convert a continuous optimization algorithm to a scheduling problem. The particle's position is a series of priority levels of assigned machines according to the order of operations. The sequence of the operations will be not changed during the iteration.

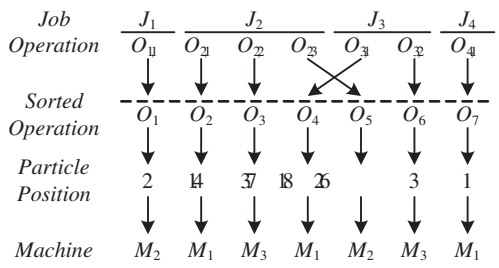

Figure 3. The Mapping between particle and FDSP.

Since the particle's position indicates the potential schedule, the position can be "decoded" to the scheduling solution. It is to be noted that the solution will be unfeasible if it violates the work-flow constraints. The operations must be started only after the completion of the previous latest operation in the work-flow. The best situation is the starting point of the operation in alignment with the ending point of its previous latest operation. After all the operations have been processed, we get the feasible scheduling solution and then calculate the cost of the solution.

4. Algorithm performance demonstration

To illustrate the effectiveness and performance of the particle swarm optimization algorithm, we will show an execution trace of the algorithm with the help of the FDSP problem involving an application with 7 operations, 3 machines and 3 data hosts represented as $(O7, M3, D3)$ problem. The speeds of the 3 machines are 4, 3, 2 CPUT, respectively, i.e. $P = \{4, 3, 2\}$. And the length of the 7 operations are 6,12,16,20,28,42,60 cycles, respectively, i.e. $L = \{6, 12, 16, 20, 28, 42, 60\}$. The flow matrix is F in

Table 1. Parameter settings for the algorithms.

Algorithm	Parameter name	value
	size of the population	20
GA	Probability of crossover	0.8
	Probability of mutation	0.09
	Swarm size	20
	Self coefficient c_1	1.49
PSO	Social coefficient c_2	1.49
	Inertia weight w	$0.9 \rightarrow 0.1$
	Clamping Coefficient ρ	0.5

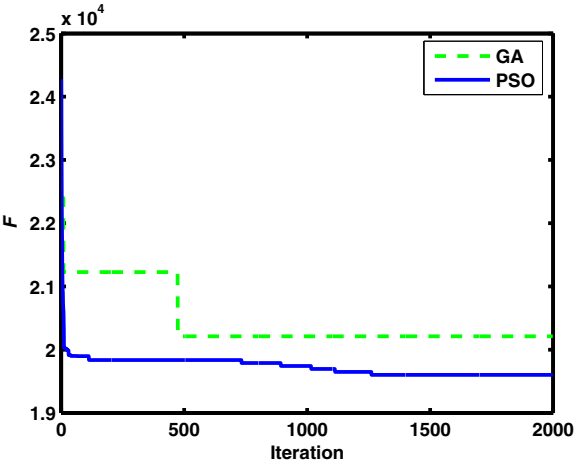

Figure 4. Performance for the FDSP $(O7, M3, D3)$

Section 2, and its other information is given as follows:

$$R = \begin{bmatrix} 6 & 18 & 76 \\ 50 & 4 & 51 \\ 1 & 85 & 15 \\ 19 & 11 & 1 \\ 39 & 12 & 0 \\ 36 & 0 & 74 \\ 61 & 82 & 30 \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 21 & 95 \\ 21 & 0 & 41 \\ 95 & 41 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 45 & 91 \\ 45 & 0 & 59 \\ 91 & 59 & 0 \end{bmatrix}$$

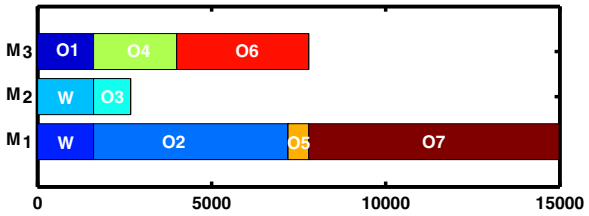

Figure 5. A scheduling solution for the FDSP $(O7, M3, D3)$

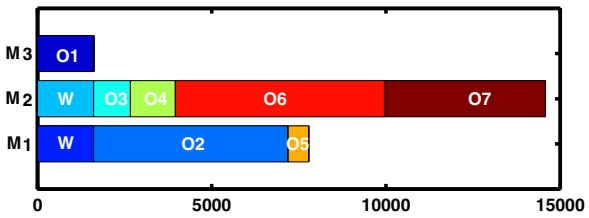

Figure 6. A scheduling solution for the FDSP $(O7, M3, D3)$

In our experiments, the algorithms used for comparison were GA (Genetic Algorithm) and PSO (Particle Swarm Optimization). The algorithms were repeated 20 times with different random seeds. Each trial had a fixed number of 2,000 iterations. Other specific parameter settings of the algorithm are described in Table 1, as recommended in [15, 16]. The fixed weights, $w_1 = w_2 = 0.5$, are used in objective cost function. The average fitness values of the best solutions throughout the optimization run were recorded. The average and the standard deviation were calculated from the 20 different trials. The standard deviation indicates the differences in the results during the 20 different trials.

Figure 4 illustrates the performance curve for our algorithm during the search processes for the FDSP problem. The results (F) for all the 20 GA runs were 20212. The best scheduling solution is $\{3, 1, 2, 3, 1, 3, 1\}$, in which the makespan is 14983 and the flowtime is 25440. Figure 5 illustrates the best scheduling solution offered by GA, in which "W" means the waiting time. The results (F) for 20 PSO runs were {19276, 19276, 19276, 20582, 19276, 19276, 19276, 19276, 20582, 20582, 19276, 19276, 19276, 19276, 19276, 19276, 20582, 19276, 19276, 20582}, with an average value of 19602. The standard deviation is 580.2057. The best scheduling solution in the 20 runs is $\{3, 1, 2, 2, 1, 2, 2\}$, in which the makespan is 14578 and the flowtime is 23973. Figure 6 shows the PSO best scheduling solution. As shown in Figure 6, the operations O_2 and O_3 both have to wait for 1611 Unit Time before they are

Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications (ISDA'06)
0-7695-2528-8/06 $20.00 © 2006 IEEE

processed in the scheduling solution. The operation O_7 is assigned to an effective machine only after all other operations had completed. So the machine M_2 has a longer work time obviously than other machines because of the work-flow constraints.

5. Conclusions

In this paper, we modeled and formulated the scheduling problem for work-flow applications in distributed data-intensive computing environments (FDSP). A particle swarm approach was proposed to solve the problem. Empirical results demonstrated that the proposed algorithm was feasible and effective. It can be applied in distributed data-intensive applications and meet the specific requirements, including work-flow constraints, data retrieval/transfer, job interact, minimum completion cost, flexibility and availability.

Our future work is targeted to generate more FDSP instances and involve more heuristics approaches.

Acknowledgments

This work was supported by NSFC (60373095) and MOST (National Strategic Basic Research '973' Program: 2005CB321904). Ajith Abraham acknowledges the support received from the International Joint Research Grant of the IITA (Institute of Information Technology Assessment) foreign professor invitation program of the Ministry of Information and Communication, South Korea.

References

[1] M. Cannataro, D. Talia, and P.K. Srimani "Parallel Data Intensive Computing in Scientific and Commercial Applications". *Parallel Computing*, 2002, 28, pp. 673–704.

[2] E. Huhlaeva, V. Kalyaevb, and N. Kruglovb. "Distributed Computing Environment for Data Intensive Tasks by Use of Metadispatcher". *Nuclear Instruments and Methods in Physics Research*, A, 2003, 502, pp. 415-C417.

[3] A. Andrieux, D. Berry, J. Garibaldi, S. Jarvis, J. MacLaren, D. Ouelhadj, and D. Snelling. "Open Issues in Grid Scheduling". *Technical Report*, Report of the workshop held at the e-Science Institute, UK, October, 21-22nd, 2003.

[4] S. Venugopal, and R. Buyya. "A Set Coverage-based Mapping Heuristic for Scheduling Distributed Data-Intensive Applications on Global Grids". *Technical Report*, GRIDS-TR-2006-3, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, March 8, 2006.

[5] N. Zhong, J. Hu, S. Motomura, J. Wu, and C. Liu. "Building A Data-mining Grid for Multiple Human Brain Data Analysis". *Computational Intelligence*, 2005, 21(2), pp. 177.

[6] F. Dong, and S.G. Akl. "Scheduling Algorithms for Grid Computing: State of the Art and Open Problems". *Technical Report*, 2006-504, School of Computing, Queen's University, Canada, January 2006.

[7] K.E. Parsopoulos, and M.N. Vrahatis. "Recent Approaches to Global Optimization Problems through Particle Swarm Optimization". *Natural Computing*, 2002, 1, pp. 235–306.

[8] J. Kennedy, and R. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, CA, 2001.

[9] M. Clerc. *Particle Swarm Optimization*. ISTE Publishing Company, London, 2006.

[10] R.C. Eberhart, and Y. Shi. "Comparison Between Genetic Algorithms And Particle Swarm Optimization". *Proceedings of IEEE International Conference on Evolutionary Computation*, 1998, pp. 611–616.

[11] D.W. Boeringer, and D.H. Werner. "Particle Swarm Optimization versus Genetic Algorithms for Phased Array Synthesis". *IEEE Transactions on Antennas and Propagation*, 2004, 52(3), pp. 771–779.

[12] A. Abraham, H. Guo, and H. Liu. "Swarm intelligence: Foundations, Perspectives And Applications". *Swarm Intelligent Systems*, Nedjah N, Mourelle L (eds.), Nova Publishers, USA, 2006.

[13] A. Salman, I. Ahmad, and S. Al-Madani. "Particle Swarm Optimization for Task Assignment Problem". *Microprocessors and Microsystems*, 2002, 26, pp. 363–371.

[14] W. Xia, and Z. Wu. "An Effective Hybrid Optimization Approach for Multi-objective Flexible Job-shop Scheduling Problems". *Computers and Industrial Engineering*, 2005, 48, pp. 409–425.

[15] M. Clerc, and J. Kennedy. "The Particle Swarm-explosion, Stability, and Convergence in A Multidimensional Complex Space". *IEEE Transactions on Evolutionary Computation*, 2002, 6, pp. 58–73.

[16] T.I. Cristian "The Particle Swarm Optimization Algorithm: Convergence Analysis and Parameter Selection". *Information Processing Letters*, 2003, 85(6), pp. 317–325.