Particle Swarm Scheduling for Work-Flow Applications in Distributed Computing Environments

Ajith Abraham[1,2], Hongbo Liu[2,3] and Mingyan Zhao[3]

[1] Centre for Quantifiable Quality of Service in Communication Systems, Faculty of Information Technology, Mathematics and Electrical Engineering, Norwegian University of Science and Technology, NO-7491 Trondheim, Norway. ajith.abraham@ieee.org, http://www.softcomputing.net
[2] School of Computer Science and Engineering, Dalian Maritime University, 116026 Dalian, China. lhb@dlut.edu.cn
[3] School of Software, Dalian University of Technology, 116620 Dalian, China. mingy_zhao@163.com

Abstract: Recently, the scheduling problem in distributed data-intensive computing environments has been an active research topic. This Chapter models the scheduling problem for work-flow applications in distributed data-intensive computing environments (FDSP) and makes an attempt to formulate the problem. Several meta-heuristics inspired from particle swarm optimization algorithm are proposed to formulate efficient schedules. The proposed variable neighborhood particle particle swarm optimization algorithm is compared with a multi-start particle swarm optimization and multi-start genetic algorithm. Experiment results illustrate the algorithm performance and its feasibility and effectiveness for scheduling work-flow applications.

1 Introduction

With the development of the high performance computing (HPC), computational grid, etc., some complex applications are designed by communities of researchers in domains such as chemistry, meteorology, high-energy physics, astronomy, biology and human brain planning (HBP) [1],[2]. For implementing and utilizing successfully those applications, one of the most important task 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 conventional situation. Scheduling jobs and resources in 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 algorithm in traditional computing paradigms barely consider the data transfer problem during mapping computational tasks, and this negligence would be costly in the case of distributed data-intensive applications [3].

Priority scheduling plays a crucial role in the differentiated services architecture for the provisioning of Quality-of-Service (QoS) of network-based applications. Jin and Min [17] proposed a novel analytical model for priority queuing systems subject to heterogeneous Long Range Dependent (LRD) self-similar or Short Range Dependent (SRD) Poisson traffic. Authors [17] applied the generalized Schilder's theorem to deal with heterogeneous traffic and further develop the analytical upper and lower bounds of the queue length distributions for individual traffic flows.

Sabrina et al. [18], discuss issues in designing resource schedulers for processing engines in programmable networks. Authors developed two CPU scheduling algorithms that could schedule CPU resource adaptively among all the competing flows. One of the packet scheduling algorithm is called start time weighted fair queueing that does not require packet processing times and the other one is called prediction based fair queueing, which uses a prediction algorithm to estimate CPU requirements of packet.

Rodrigues et al. [19] proposed a branch and bound approach based on constraint-based search (CBS) for scheduling of continuous processes. Tasks time-windows are submitted to a constraint propagation procedure that identifies existing orderings among tasks and linear programming is used to determine the optimal flow rate for each bucket whenever all buckets are ordered in the branch and bound.

In this chapter, we introduce the scheduling problem for work-flow applications in distributed data-intensive computing environments. Rest of the Chapter is organized as follows. We model and formulate the problem in Section 2. We present an approach based on particle swarm algorithm based heuristics in Section 3. In Section 4, experiment results and discussions are provided. Finally, we conclude our work in the chapter.

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 these technical terms are neither clearly stated nor consistently used by different researchers, which frequently makes readers confused. For clarity purposes, some key terminologies are re-defined for formulating the problem.

- 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 supercomputer, or a cluster of workstations. The computational capacity of the machine is depend 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 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 workflow 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 dependencies 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 as "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 considered, the processing speeds 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 $\{O_1, O_2, \cdots, O_n\}$. The processing lengths of the operation are $\{L_1, L_2, \cdots, L_n\}$. All the operations are in the specific work-flow, and they will be carried orderly on the machines with data retrieval, data input and data output.

The operations in the work-flow can be represented as or transformed to a Directed Acyclic Graph (DAG), where each node in the DAG represents an operation and the edges denote control/data dependencies. 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 of 9 operations. The recursive loop between O_1 and O_9 can be neglected when the scheduling focus on the stage within the loop. Its flow matrix F is represented as follows:

$$\begin{bmatrix} -1 & 8 & 3 & 9 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & 5 & 6 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & 2 & 12 & 11 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & 7 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 13 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 4 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 8 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

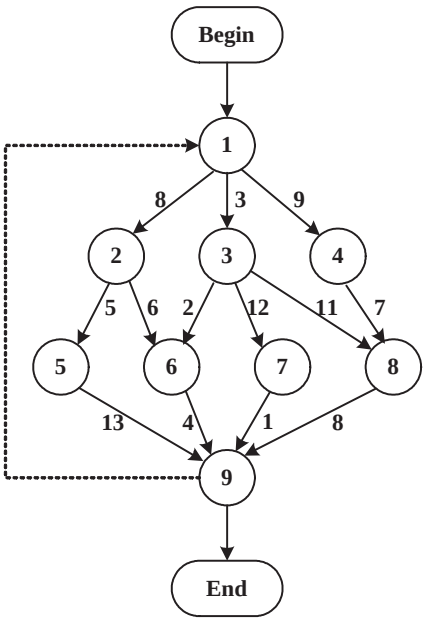

Fig. 1. A works-flow application with 9 operations.

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 matrices $A = [a_{i,j}]$ and $B = [b_{i,j}]$, where the element $a_{i,j}$ in the former is the distance between 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} finishes the operation O_i. For the operation O_i, its completion time C_{O_i} can be calculated by Eq. (1).

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

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 aggregated as a weighted combination:

$$F = w_1 min\{F_1\} + w_2 min\{F_2\} \tag{2}$$

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 article. In fact, the dynamic weighted aggregation mainly takes C_{max} into account [4] because C_{sum} is commonly much larger than C_{max} and the solution has a large weight on C_{sum} during minimizing of the objective. Alternatively, the weights can be changed gradually according to the Eqs. (3) and (4). The changes in the dynamic weights ($R = 200$) are illustrated in Figure 2.

$$w_1(t) = |sin(2\pi t/R)| \tag{3}$$

$$w_2(t) = 1 - w_1(t) \tag{4}$$

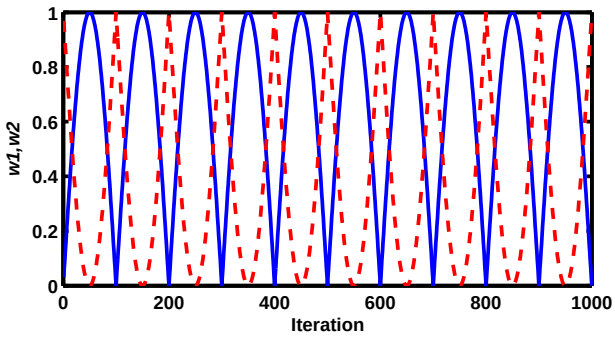

Fig. 2. Changes in dynamic weights.

3 Particle Swarm Heuristics for FDSP

3.1 Canonical Model

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 [5, 6]. 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 global optimization algorithms [7, 8, 9]. The canonical PSO 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 $\mathbf{x}_i$ (i is the index of the particle), and a velocity represented by a velocity-vector $\mathbf{v}_i$. Each particle remembers its own best position so far in a vector $\mathbf{x}_i^{\#}$, and its j-th dimensional value is $x_{ij}^{\#}$. The best position-vector among the swarm so far is then stored in a vector $\mathbf{x}^*$, and its j-th dimensional value is x_j^*. During the iteration time t, the update of the velocity from the previous 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).

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

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

where w is called as the inertia factor, r_1 and r_2 are the random numbers, which are used to maintain the diversity of the population, and are uniformly distributed in the interval [0,1] for the j-th dimension of the i-th particle. c_1 is a positive constant, called as coefficient of the self-recognition component, c_2 is a positive constant, called as coefficient of the social component.

From Eq.(5), a particle decides where to move next, considering its own experience, which is the memory of its best past position, and the experience of its most successful particle in the swarm. In the particle swarm model, the particle searches the solutions in the problem space with 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 must be clamped in between the maximum velocity $[-v_{max}, v_{max}]$ given in Eq.(7):

$$v_{ij} = sign(v_{ij})min(|v_{ij}|, v_{max}). \quad (7)$$

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

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

Algorithm 1 Particle Swarm Optimization Algorithm

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

The termination criteria are usually one of the following:

- Maximum number of iterations: the optimization process is terminated after a fixed number of iterations, for example, 1000 iterations.
- Number of iterations without improvement: the optimization process is terminated after some fixed number of iterations without any improvement.
- Minimum objective function error: the error between the obtained objective function value and the best fitness value is less than a pre-fixed anticipated threshold.

The role of inertia weight w, in Eq.(5), is considered critical for the convergence behavior of PSO. The inertia weight is employed to control the impact of the previous history of velocities on the current one. Accordingly, the parameter w regulates the trade-off between the global (wide-ranging) and local (nearby) exploration abilities of the swarm. A large inertia weight facilitates global exploration (searching new areas), while a small one tends to facilitate local exploration, i.e. fine-tuning the current search area. A suitable value for the inertia weight w usually provides balance between global and local exploration abilities and consequently results in a reduction of the number of iterations required to locate the optimum solution. Initially, the inertia weight is set as a constant. However, some experiment results indicates that it is better to initially set the inertia to a large value, in order to promote global exploration of the search space, and gradually decrease it to get more refined solutions [10, 11]. Thus, an initial value around 1.2 and gradually reducing towards 0 can be considered as a good choice for w. A better method is to use some adaptive approaches (example: fuzzy controller), in which the parameters can be adaptively fine tuned according to the problem under consideration [12, 13].

The parameters c_1 and c_2, in Eq.(5), are not critical for the convergence of PSO. However, proper fine-tuning may result in faster convergence and alleviation of local minima. As default values, usually, $c_1 = c_2 = 2$ are used, but some experiment results indicate that $c_1 = c_2 = 1.49$ might provide even better results. Recent work reports that it might be even better to choose a larger cognitive parameter, c_1, than a social parameter, c_2, but with $c_1 + c_2 \leq 4$ [16].

The particle swarm algorithm can be described generally as a population of vectors whose trajectories oscillate around a region which is defined by each individual's previous best success and the success of some other particle. Various methods have been used to identify some other particle to influence the individual. Eberhart and Kennedy called the two basic methods as "gbest model" and "lbest model" [5]. In the lbest model, particles have information only of their own and their nearest array neighbors' best (lbest), rather than that of the entire group.

In the gbest model, the trajectory for each particle's search is influenced by the best point found by any member of the entire population. The best particle acts as an attractor, pulling all the particles towards it. Eventually all particles

will converge to this position. The lbest model allows each individual to be influenced by some smaller number of adjacent members of the population array. The particles selected to be in one subset of the swarm have no direct relationship to the other particles in the other neighborhood.

Typically lbest neighborhoods comprise exactly two neighbors. When the number of neighbors increases to all but itself in the lbest model, the case is equivalent to the gbest model. Some experiment results testified that gbest model converges quickly on problem solutions but has a weakness for becoming trapped in local optima, while lbest model converges slowly on problem solutions but is able to "flow around" local optima, as the individuals explore different regions. The gbest model has faster convergence. But very often for multi-modal problems involving high dimensions it tends to suffer from premature convergence.

3.2 Variable Neighborhood Particle Swarm Optimization Algorithm (VNPSO)

Variable Neighborhood Search (VNS) is a relatively recent metaheuristic which relies on iteratively exploring neighborhoods of growing size to identify better local optima with shaking strategies [14, 15]. More precisely, VNS escapes from the current local minimum x^* by initiating other local searches from starting points sampled from a neighborhood of x^*, which increases its size iteratively until a local minimum is better than the current one is found. These steps are repeated until a given termination condition is met. The metaheuristic method, Variable Neighborhood Particle Swarm Optimization (VNPSO) algorithm, was originally inspired by VNS [20]. In PSO, if a particle's velocity decreases to a threshold v_c, a new velocity is assigned using Eq.(9):

$$v_{ij}(t) = w\hat{v} + c_1 r_1(x_{ij}^{\#}(t-1) - x_{ij}(t-1)) + c_2 r_2(x_j^*(t-1) - x_{ij}(t-1)) \quad (9)$$

$$\hat{v} = \begin{cases} v_{ij} & \text{if } |v_{ij}| \geq v_c \\ u(-1,1)v_{max}/\eta & \text{if } |v_{ij}| < v_c \end{cases} \quad (10)$$

The VNPSO algorithm scheme is summarized as Algorithm 2. The performance of the algorithm is directly correlated to two parameter values, v_c and η. A large v_c shortens the oscillation period, and it provides a great probability for the particles to leap over local minima using the same number of iterations. But a large v_c compels the particles in the quick "flying" state, which leads them not to search the solution and forcing them not to refine the search. The value of η changes directly the variable search neighborhoods for the particles. It is to be noted that the algorithm is different from the multi-start technique. We also implemented the Multi-Start Particle Swarm Optimization (MSPSO) (illustrated in Algorithm 3) and the Multi-Start Genetic Algorithm (MSGA) to compare the empirical performances.

Algorithm 2 Variable Neighborhood Particle Swarm Optimization

01. Initialize the size of the particle swarm n, and other parameters.
02. Initialize the positions and the velocities for all the particles randomly.
03. Set the flag of iterations without improvement $Nohope = 0$.
04. While (the end criterion is not met) do
05. $t = t + 1$;
06. Calculate the fitness value of each particle;
07. $\mathbf{x}^* = argmin_{i=1}^{n}(f(\mathbf{x}^*(t-1)), f(\mathbf{x}_1(t)), f(\mathbf{x}_2(t)), \cdots, f(\mathbf{x}_i(t)), \cdots, f(\mathbf{x}_n(t)))$;
08. If $\mathbf{x}^*$ is improved then $Nohope = 0$, else $Nohope = Nohope + 1$.
09. For i= 1 to n
10. $\mathbf{x}_i^{\#}(t) = argmin_{i=1}^{n}(f(\mathbf{x}_i^{\#}(t-1)), f(\mathbf{x}_i(t))$;
11. For $j = 1$ to d
12. If $Nohope < 10$ then
13. Update the j-th dimension value of $\mathbf{x}_i$ and $\mathbf{v}_i$
14. according to Eqs.(5),(7),(6),(8);
15. else
16. Update the j-th dimension value of $\mathbf{x}_i$ and $\mathbf{v}_i$
17. according to Eqs.(10),(9),(6),(8).
18. Next j
19. Next i
20. End While.

Algorithm 3 Multi-start Particle Swarm Optimization

01. Initialize the size of the particle swarm n, and other parameters.
02. Initialize the positions and the velocities for all the particles randomly.
03. Set the flag of iterations without improvement $Nohope = 0$.
04. While (the end criterion is not met) do
05. $t = t + 1$;
06. Calculate the fitness value of each particle;
07. $\mathbf{x}^* = argmin_{i=1}^{n}(f(\mathbf{x}^*(t-1)), f(\mathbf{x}_1(t)), f(\mathbf{x}_2(t)), \cdots, f(\mathbf{x}_i(t)), \cdots, f(\mathbf{x}_n(t)))$;
08. If $\mathbf{x}^*$ is improved then $Nohope = 0$, else $Nohope = Nohope + 1$.
09. For i= 1 to n
10. $\mathbf{x}_i^{\#}(t) = argmin_{i=1}^{n}(f(\mathbf{x}_i^{\#}(t-1)), f(\mathbf{x}_i(t))$;
11. For $j = 1$ to d
12. If $Nohope < 10$ then
13. Update the j-th dimension value of $\mathbf{x}_i$ and $\mathbf{v}_i$
14. according to Eqs.(5),(7),(6),(8);
15. else
16. Re-initialize the positions and the velocities
17. for all the particles randomly.
18. Next j
19. Next i
20. End While.

For applying PSO successfully for the FDSP problem, one of the key issues is the mapping of the problem solution to the PSO particle space, which directly affects its feasibility and performance. We setup a search space of n dimension for an $(n - Operations, m - Machines)$ FDSP problem. Each dimension was limited to $[1, m + 1)$. For example, consider a little scale $(7 - Operations, 3 - Machines)$ FDSP, Fig. 3 shows a mapping between a one possible assignment instance to a particle position coordinates in the PSO 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 to during the course of PSO. So the value of a particle's position should be an 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.

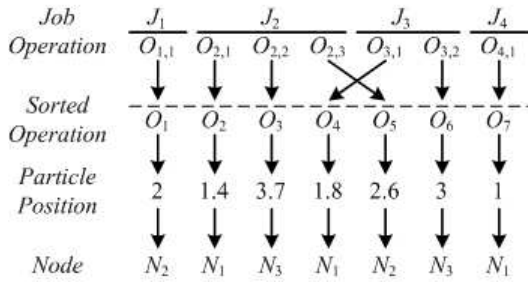

Fig. 3. The Mapping between particle and FJSP.

Since the particle's position indicates the potential schedule, the position can be "decoded" to the feasible solution. It is to be noted that the position matrix may violate the work-flow constraints. The starting point of 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 Experiment Results and Algorithm Performance Demonstration

To illustrate the effectiveness and performance of the particle swarm searching algorithm, three representative instances based on practical data have been selected. In our experiments, the algorithms used for comparison were VNPSO, MSPSO (Multi-start PSO) and MSGA (Multi-start GA). In VNPSO, η and v_c were set to 2 and 1e-7 before 15,000 iterations, while they were set to 5 and 1e-10 after 15,000 iterations. Other specific parameter settings of the different algorithms are described in Table 1. The algorithms were run 20 times with different random seeds. Each trial had a fixed number of 2,000 iterations. The average fitness values of the best solutions throughout the optimization run were recorded. Usually another emphasis will be to generate the schedules at a minimal amount of time. So the completion time for 20 trials were used as one of the criteria to improve their performance.

Table 1. Parameter settings for the algorithms.

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

We illustrate a small scale FDSP problem involving an application with 9 operations, 3 machines and 3 data hosts represented as $(O9, M3, D3)$ problem. The speeds of the 3 machines are 4, 3, 2 CPUT, respectively, i.e., $P = \{4, 3, 2\}$. The length of the 9 operations are 6,12,16,20,28,36,42,52,60 cycles, respectively, i.e., $L = \{6, 12, 16, 20, 28, 36, 42, 52, 60\}$. The flow matrix is F as depicted in Section 2, and all other information are as follows:

$$R = \begin{bmatrix} 6 & 18 & 76 \\ 50 & 4 & 51 \\ 1 & 85 & 15 \\ 19 & 11 & 1 \\ 39 & 12 & 0 \\ 73 & 0 & 1 \\ 57 & 29 & 77 \\ 36 & 0 & 74 \\ 61 & 82 & 30 \end{bmatrix}$$

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

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

Figure 4 illustrates the performance of the three algorithms during the search processes for $(O9, M3, D3)$ problem. The best scheduling solution in the 20 MSGA runs is $\{3, 1, 2, 3, 1, 1, 1, 3, 1\}$, in which the makespan is 23654 and the flowtime is 34075.

The best scheduling solution obtained in the 20 MSPSO and VNPSO runs is $\{3, 1, 2, 3, 1, 1, 2, 3, 2\}$, in which the makespan is 15011 and the flowtime is 30647. While MSPSO provides the best results 8 times in 20 runs, VNPSO provides the best result 10 times in the same runs respectively.

Figure 5 provides an optimal schedule for $((O9, M3, D3)$ problem, in which "W" means the waiting time. As depicted in 5, the operations O_2 and O_3 both have to wait for 1611 time units before they are processed in the scheduling solution.

Further, we tested the three algorithms for two more FDSP problems, i.e. $(O10, M3, D3)$ and $(O12, M4, D3)$. Empirical results are illustrated in Table 2. In general, VNPSO performs better than the other two approaches, although its computational time is worse than MSPSO. VNPSO could be an ideal approach for solving the large scale problems when other algorithms failed to give a better solution.

Table 2. Comparison of performance for different FDSPs.

Instance	*Items*	MSGA	MSPSO	VNPSO
$(O9, M3, D3)$	average	28864	24152	28.8000
	time	200.2780	133.6920	181.2970
$(O10, M3, D3)$	average	21148	19594	16389
	time	210.6230	138.5890	140.3920
$(O12, M4, D3)$	average	16146	14692	14412
	time	235.1080	152.5520	154.4420

5 Conclusions

In this chapter, we modeled and formulated the scheduling problem for work-flow applications in distributed data-intensive computing environments (FDSP). A particle swarm optimization based variable neighborhood search

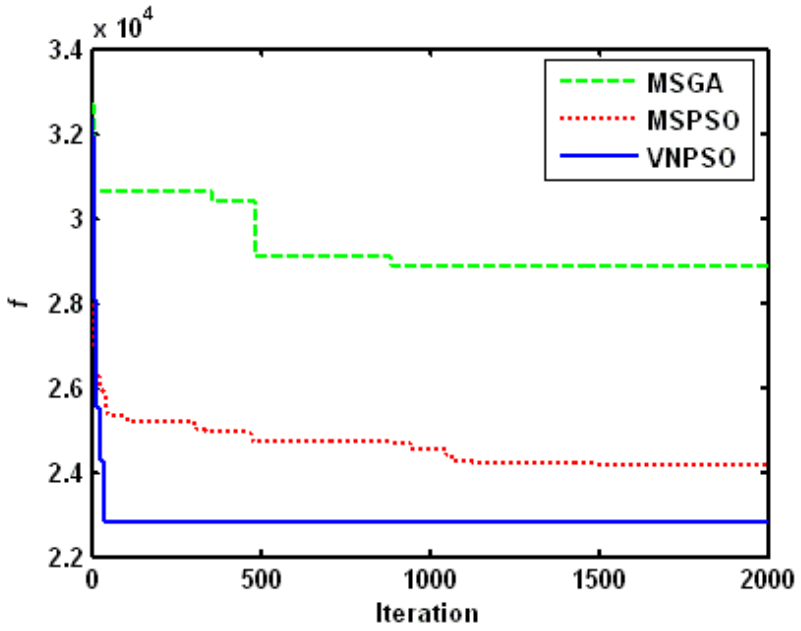

Fig. 4. Performance for the FDSP $(O9, M3, D3)$

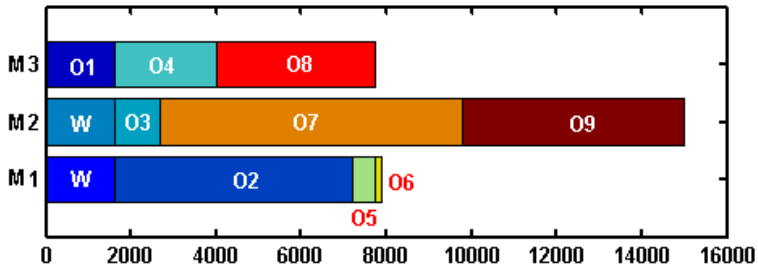

Fig. 5. A scheduling solution for the FDSP $(O9, M3, D3)$

is proposed to solve the problem. Empirical results demonstrate that the proposed VNPSO algorithm is feasible and effective. VNPSO can be applied in distributed data-intensive applications to meet the specified requirements, including work-flow constraints, data retrieval/transfer, job interaction, minimum completion cost, flexibility and availability.

Our future research is targeted to generate more FDSP instances and investigate more optimization/meta-heuristic approaches.

6 Acknowledgements

This work was partly supported by NSFC (60373095), DLMU (DLMU-ZL-200709) and the Research Council, NTNU and UNINETT.

References

1. I. Foster and C. Kesselman (Eds.) " The Grid: Blueprint for a New Computing Infrastructure". *Morgan-Kaufmann*, 1998.
   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.
2. 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.
3. 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.
4. K.E. Parsopoulos, and M.N. Vrahatis. "Recent Approaches to Global Optimization Problems through Particle Swarm Optimization". *Natural Computing*, 2002, 1, pp. 235–306.
5. J. Kennedy, and R. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, CA, 2001.
6. M. Clerc. *Particle Swarm Optimization*. ISTE Publishing Company, London, 2006.
7. 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.
8. 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.
9. 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.
10. J. Kennedy J and R. Mendes. "Population structure and particle swarm performance". *Proceeding of IEEE conference on Evolutionary Computation*, 2002, pp. 1671–1676.
11. H. Liu, B. Li, Y. Ji and T. Sun. "Particle Swarm Optimisation from lbest to gbest". *Applied Soft Computing Technologies: The Challenge of Complexit*, Springer Verlag, 2006, pp. 537–545.
12. Y. H. Shi and R. C. Eberhart. "Fuzzy adaptive particle swarm optimization". *Proceedings of IEEE International Conference on Evolutionary Computation*, 2001, pp. 101–106.
13. H. Liu and A. Abraham. "Fuzzy Adaptive Turbulent Particle Swarm Optimization". *Proceedings of the Fifth International conference on Hybrid Intelligent Systems*, 2005, pp. 445–450.

14. P. Hansen and N. Mladenović N. "Variable neighbourhood search:Principles and applications". *European Journal of Operations Research*, 2001, 130, pp. 449–467.
15. P. Hansen and N. Mladenović N. "Variable neighbourhood search". *Handbook of Metaheuristics*, Dordrecht, Kluwer Academic Publishers, 2003.
16. 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.
17. X. Jin and G. Min, Performance analysis of priority scheduling mechanisms under heterogeneous network traffic Journal of Computer and System Sciences, Volume 73, Issue 8, pp. 1207-1220, 2007.
18. F. Sabrina, C.D. Nguyen, S. Jha, D. Platt and F. Safaei, Processing resource scheduling in programmable networks Computer Communications, Volume 28, Issue 6, pp. 676-687, 2005.
19. L.C.A. Rodrigues, R. Carnieri and F. Neves Jr., Scheduling of continuous processes using constraint-based search: An application to branch and bound Computer Aided Chemical Engineering, Volume 10, pp. 751-756, 2002.
20. A. Abraham, H. Liu, and T.G. Chang, Variable Neighborhood Particle Swarm Optimization Algorithm, Genetic and Evolutionary Computation Conference (GECCO-2006), Seattle, USA, 2006.

Index