Scheduling Data-Intensive Work-Flow Applications Using Particle Swarm Approaches

HONGBO LIU¹, AJITH ABRAHAM², OKKYUNG CHOI²,³, SEONG HWAN MOON ³

¹Department of Computer Science, Dalian University of Technology
Linggong Road 2, Dalian 116023, CHINA
²School of Computer Science and Engineering, Chung-Ang University
221, Heukseok-dong, Dongjak-gu, Seoul 156-756, KOREA
³Department of Science and Technology, Education for Life
Seoul National University of Education, Seocho-dong
Seocho-gu, Seoul 137-742, KOREA

Abstract: Data-intensive work-flow applications increase continuously in various domains. The job scheduling problem usually has to take into account of the computational loads at each computing resource, the distributions of data required by each job, and the work-flow constraints. The scheduling problem is one of the major difficult tasks in these types of distributed computing environments. This paper formulates the scheduling problem for data-intensive work-flow applications (DFSP) and solve the problem using particle swarm optimization approaches. The details of implementation for DFSP are provided and the corresponding computational experiments are reported. The results indicate that the proposed algorithm is an efficient approach for DFSP.

Key–Words: Data-intensive, Work-flow, Particle swarm algorithm, Genetic algorithm, Variable Neighborhood Search

1 Introduction

Some new complex applications are deployed by communities of researchers in various domains such as high-energy physics, astronomy, biology [1] and Human Brain Planning (HBP) [2]. For implementing and utilizing successfully these applications, one of the most important activity is to find appropriate schedules before the application is executed [3, 4]. The goal is to find an optimal assignment of tasks for the applications with respect to the costs of the available resources. However, computing resources and data resources are geographically distributed under different ownerships each having their own access policy, cost and various constraints. 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 neglect will be costly in distributed data-intensive applications [5]. The paper formulates the new scheduling problem, which takes into account of the work-flow constraints, components interactions and data retrieval. The scheduling problem is known to be NP-complete [6].

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 [7]. It could be implemented and applied easily to solve various function optimization problems, or the problems that can be transformed to function optimization problems. The particle swarm algorithms are extended to solve the problem. Rest of the paper is organized as follows. The objective problem is formulated 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

To formulate the scheduling problem, suppose a work-flow application comprises of q Jobs {J₁, J₂, ⋅⋅⋅, J₉}, m Machines {M₁, M₂, ⋅⋅⋅, M₉} and k Data hosts {D₁, D₂, ⋅⋅⋅, D₉}. In the application, the calculating speed of the machine are {P₁, P₂, ⋅⋅⋅, P₉}. Each job consists of a set of operations Jⱼ = {O₁,j, O₂,j, ⋅⋅⋅, Oₙ,j}. For convenience, all the jobs will be decomposed to atomic operations
and the operations are re-sorted as \( \{O_1, O_2, \cdots, O_n\} \). Their processing lengths are \( L_1, L_2, \cdots, L_n \), respectively. When the operations are assigned to the machines to be executed, all the operations are in the specific work-flow. In other words, they will be carried orderly out on the machines. 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. 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 9 operations. The recursive loop between \( O_1 \) and \( O_9 \) can be neglected when the scheduling focused on the stage within the loop. Its flow matrix \( F \) is

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

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 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 \( S_i \) completes the operation \( O_i \). For the operation \( O_i \), its completion time \( C_{O_i} \) can be calculated by Eq. (1).

\[
C_{O_i} = \sum_{l=1}^{n} f_{i,l} a_{S_i,l} + \sum_{k=1}^{n} r_{i,k} b_{S_i,k} + L_i/P_{S_i}
\]

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 \( C_{\text{max}} = \max\{\sum C_{M_i}\} \) as the makespan, and \( C_{\text{sum}} = \sum_{i=1}^{n}\sum C_{M_i} \) as the flowtime. The scheduling problem minimizes the maximum completion time (makespan: \( C_{\text{max}} \)) and the sum of the completion times (flowtime: \( C_{\text{sum}} \)).

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

\[
f = w_1 \min\{F_1\} + w_2 \min\{F_2\}
\]

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.

3 Particle swarm heuristics for DFSP

Particle Swarm Optimization (PSO) is a population-based optimization tool [8]. As an algorithm, its main
strength is its fast convergence, which compares favorably with many other global optimization algorithms [9, 10, 11]. 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 $\vec{x}^\ast$, and its j-th dimensional value is $x^\ast_{ij}$. During the iteration time $t$, the update of the velocity from the previous velocity to the new velocity is determined by Eq.(3). The new position is then determined by the sum of the previous position and the new velocity by Eq.(4).

$$v_{ij}(t+1) = w v_{ij}(t) + c_1 r_1 (x^\#_{ij}(t) - x_{ij}(t)) + c_2 r_2 (x^\ast_{ij}(t) - x_{ij}(t))$$ (3)

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1)$$ (4)

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.(5), and similarly for its moving range given in Eq.(6):

$$v_{i,j} = \text{sign}(v_{i,j}) \min(|v_{i,j}|, v_{max})$$ (5)

$$x_{i,j} = \text{sign}(x_{i,j}) \min(|x_{i,j}|, x_{max})$$ (6)

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. $\rho = 1$.

For applying particle swarm algorithm successfully for the DFSP problem, one of the key issues is how to map the problem solution to the particle space, which directly affects its feasibility and performance [12, 13]. We setup a search space of n dimension for an $(n - \text{Operations}, m - \text{Machines})$ DFSP problem. Each dimension is limited to $[1, m + 1]$. For example, consider the $(7 - \text{Operations}, 3 - \text{Machines})$ DFSP. Figure 2 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

Figure 2: The Mapping between particle and DFSP.
ing strategy and get them to explore variable neighborhood spaces for the better scheduling solutions. In PSO, if a particle’s velocity decreases to a threshold \( v_c \), a new velocity is assigned using Eq.(7):

\[
v_{ij}(t + 1) = w \dot{v}_{ij} + c_1 r_1(x^*_{ij}(t) - x_{ij}(t)) + c_2 r_2(x^*_j(t) - x_{ij}(t))
\]

\[
\dot{v} = \begin{cases} 
  v_{ij} & \text{if } |v_{ij}| \geq v_c \\
  u(-1, 1)v_{\text{max}}/\eta & \text{if } |v_{ij}| < v_c
\end{cases}
\]

Algorithm 1 VNPSO

01. Initialize the size of the particle swarm \( n \), and other parameters; Initialize the positions and the velocities for all the particles randomly.
02. Set the flag of iterations without improvement \( N_{\text{nohope}} = 0 \).
03. While (the end criterion is not met) do
04. \( t = t + 1 \);
05. Calculate the fitness value of each particle;
06. If \( \bar{x}^* \) is improved then \( N_{\text{nohope}} = 0 \), else \( N_{\text{nohope}} = N_{\text{nohope}} + 1 \).
07. For \( i = 1 \) to \( n \)
08. For \( j = 1 \) to \( \text{Dimension} \)
09. \( \bar{x}^*_i(t) = \arg\min_{\bar{x}_i(t-1)}(f(\bar{x}^*_{i}(t-1)), f(\bar{x}^*_i(t))) \)
10. End For
11. End For
12. End While.

Algorithm 2 MSPSO

01. Initialize the size of the particle swarm \( n \), and other parameters; Initialize the positions and the velocities for all the particles randomly.
02. Set the flag of iterations without improvement \( N_{\text{nohope}} = 0 \).
03. While (the end criterion is not met) do
04. \( t = t + 1 \);
05. Calculate the fitness value of each particle;
06. If \( \bar{x}^* \) is improved then \( N_{\text{nohope}} = 0 \), else \( N_{\text{nohope}} = N_{\text{nohope}} + 1 \).
07. For \( i = 1 \) to \( n \)
08. For \( j = 1 \) to \( \text{Dimension} \)
09. \( \bar{x}^*_i(t) = \arg\min_{\bar{x}_i(t-1)}(f(\bar{x}^*_{i}(t-1)), f(\bar{x}^*_i(t))) \)
10. End For
11. End For
12. End While.

VNPSO algorithm scheme is summarized as Algorithm 1. The performance of the algorithm is directly correlated to two parameter values, \( v_c \) and \( \eta \). 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 \( \eta \) changes directly the variable search neighborhoods for the particles. It is to be noted that the algorithm is different from the multi-start technique, shown in Algorithm 2. We also implemented the Multi-Start PSO (MSPSO) and Multi-Start GA (MSGA) to compare their performances.

4 Algorithm performance demonstration

To illustrate the effectiveness and performance of the proposed algorithm, we select the representative instances (problem \( O9, M3, D3 \)) to compute. In our experiments, the algorithms used for comparison were MSGA (Multi-start Genetic Algorithm), MSPSO (Multi-start PSO) and VNPSO (Variable Neighborhood PSO). The algorithms were repeated 20 times with different random seeds. Each trial had a fixed number of 2,000 iterations. In VNPSO, \( \eta \) and \( v_c \) were set to 2 and 1e-7 before 2,000 iterations, while they were set to 5 and 1e-10 after 1,500 iterations. \( v_{\text{max}} \) was set to \( s/2 \), the range of the position domain. The fixed weights, \( w_1 = w_2 = 0.5 \), are used in objective cost function. Other specific parameter settings of the algorithm are described in Table 1, as recommended in [15]. The average fitness values of the best solutions throughout the optimization run were recorded. The averages \( \langle f \rangle \) was calculated from the
20 different trials. 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.

We will show an execution trace of the algorithm with the DFSP problem \((O9, M3, D3)\) 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. \(P = \{4, 3, 2\}\). And 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 \(F\) in Section 2, and its other information is given as follows:

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

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

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

Figure 3 illustrates the performance curves for the considered algorithms during the search processes for the \((O9, M3, D3)\) DFSP problem. The results \((f)\) for all the 20 MSGA runs were 28864 and the computing time is 200.2780. The best scheduling solution is \(\{3, 1, 2, 3, 1, 1, 3, 1, 1\}\), in which the makespan is 23654 and the flowtime is 34075. The results \((f)\) for 20 MSPSO runs were \{24243, 22829, 22829, 24243, 22829, 26724, 22829, 23996, 23996, 22829, 22829, 24243, 26304, 27702, 23996, 22829, 22829, 26724, 24243, 23996\}, with an average value of 24152. The computing time is 133.6920. The results \((f)\) for 20 VNPSO runs were \{22829, 23996, 22829, 23996, 22829, 23996, 25520, 24243, 22829, 22829, 22829, 22829, 23996, 23996, 22829, 23996, 23996, 22829, 22829\}, with an average value of 23443. The computing time is 134.8140. The best scheduling solution that MSPSO and VNPSO provided in the 20 run is \(\{3, 1, 2, 3, 1, 1, 2, 3, 2\}\), in which the makespan is 15011 and the flowtime is 30647, as showed in Figure 4. But MSPSO provided the best scheduling solution eight time while VNPSO provided the best scheduling solution ten time. In Figure 4, “W” means the waiting time. As shown in Figure 4, the operations \(O_2\) and \(O_3\) both have to wait for 1611 Unit Time before they are processed in the scheduling solution. The operation \(O_9\) 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. VNPSO performs better than the other two approaches, although its time item is slightly worse than MSPSO. VNPSO could be an ideal approach for solving the DFSP problem when other algorithms failed to give a better solution.

### 5 Conclusions

In this paper, we formulated the scheduling problem for work-flow applications in distributed data-intensive computing environments (DFSP). A variable neighborhood particle swarm optimisation approach was proposed to solve the problem. Empirical results...
demonstrated that our 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. The proposed approach outperformed the MSGA (Multi-start Genetic Algorithm) approach by providing the best scheduling solution and MSPSO (Multi-start PSO) in terms of stability.

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

Acknowledgements: This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD, Basic Research Promotion Fund) (KRF-2006-353-B00059).

References:


