Multiobjective Differential Evolution for Workflow Execution on Grids

A.K.M. Khaled Ahsan Talukder, Michael Kirley and Rajkumar Buyya
Dept. of Computer Science and Software Engineering
The University of Melbourne
Victoria 3053, Australia
{akmkat,mkirley,raj}@csse.unimelb.edu.au

ABSTRACT

Most algorithms developed for scheduling applications on global Grids focus on a single Quality of Service (QoS) parameter such as execution time, cost or total data transmission time. However, if we consider more than one QoS parameter (eg. execution cost and time may be in conflict) then the problem becomes more challenging. To handle such scenarios, it is convenient to use heuristics rather than a deterministic algorithm. In this paper we have proposed a workflow execution planning approach using Multiobjective Differential Evolution (MODE). Our goal was to generate a set of trade-off schedules according to two user specified QoS requirements (time and cost). The alternative trade-off solutions offer more flexibility to users when estimating their QoS requirements of workflow executions. We have compared our results with two baseline multiobjective evolutionary algorithms. Simulation results show that our modified MODE is able to find a comparatively better spread of compromise solutions.

Categories and Subject Descriptors

I.2.8 [**Problem Solving,Control Methods and Search**]: Scheduling; C.2.4 [**Distributed Systems**]: Distributed applications

General Terms

Grid Scheduling, Multiobjective Optimization, Multiobjective Differential Evolution

1. INTRODUCTION

Grid Computing aims to allow unified access to data, computing power, sensors and others resources through a single virtual laboratory [4]. Technologies should provide services, protocols and software needed for the flexible and controlled sharing of resources. However, such infrastructure has to be competitive in terms of costs to the final users. Usually this corresponds to identifying the optimal use of the resources.

There are basically two approaches to solve this optimization problem. The first is based on a distributed resources discovery and allocation system. The second is based on a central repository of resources and resources requests. The first is suitable for small jobs that can easily be accepted by the computing Grid. The second is suitable for large periodical jobs that are scheduled in advance. In this case, the centralized scheduler is usually referred to as "metascheduler", because of its position on top of the local schedulers. It should optimize the allocation of a job allowing the execution on the "fittest" set of resources. In most cases, such as in scientific and enterprize domains, a typical application is constructed as workflows. Here, the metascheduler will deploy the workflow on the Grid considering the current availability of resources, time constraints and user specified QoS.

A number of Grid workflow management systems [3] have been developed to facilitate the composition and execution of workflow applications over distributed resources. Many heuristic [5],[13],[14] have also been proposed for workflow scheduling in order to optimize a single objective, such as minimizing execution time. However, a large number of objectives need to be considered when scheduling workflows on utility Grids based on users QoS requirements. In this paper, we introduce a model that can optimize conflicting objectives. However, in many Differential Evolution based Job-Shop/Flow-Shop scheduling, the algorithm was implemented by mapping Job/Machine sequence to real numbers. Since this approach is not feasible in the case of Grid scheduling, we have to deal with exact scheduling sequences and thus our approach does not need to map the resource/task sequence to real values as described in [8] and [9].

2. MOTIVATION AND RELATED WORK

Scheduling using nature's heuristics is not a new idea. There are numerous studies reporting work done on scheduling DAG (Directed Acyclic Graph) based task graphs in multiprocessor systems [13]. Genetic algorithms and HEFT (Heterogeneous Earliest Finish Time) have been extended by the ASKALON project [10] to schedule scientific applications in Grid environments. Recently in [17], a different base-line algorithm for Multiobjective Optimization was tested on different workflow model.

In a similar study [16], their model was compared with Max-Min and Min-Min heuristics described in [5]. There are also examples of scheduling using differential evolution(DE) [8],[9] where the chromosome represents a feasible schedule that needs to be converted to a string of real numbers for DE operations. The real numbers can be inverse mapped to a schedule. The forward and inverse mapping may lead to creation of infeasible solutions. The work in this paper is distinct from the related work because it simultaneously optimizes multiple objectives of workflow execution according to users' QoS without the unwanted creation of infeasible solutions.

3. PROBLEM OVERVIEW

We formalized the workflow planning problem as a multiobjective optimization problem (MOP) in which a group of conflicting objectives are simultaneously optimized. As given in equation (1) and (2), there are two conflicting objectives: minimize execution time and minimize execution cost. In such problems, there is no single optimal solution but rather a set of potential solutions. We can define a multiobjective problem as

$$minimize \quad f(x) = f_1(x), f_2(x), \ldots, f_k(x) \tag{1}$$

where $x \epsilon X$ and X is a solution space. A solution is said to dominate another solution if it is as good as the other and better in at least one objective. That is x^* dominates x, if and only if

$$\forall i \epsilon [1, 2, \ldots, k], f_i(x^*) \leq f_i(x) \wedge \exists j \epsilon [1, 2, \ldots, k], f_j(x^*) < f_j(x) \tag{2}$$

The resource scheduling problem for the Grid consists of arranging the pairs of jobs and resources given certain constraints. The objective is to minimize the mult-dimensional QoS metrics. We regard completion time of jobs and total execution cost of jobs as multi-dimensional QoS metrics. To address the problem, we have to start with the following assumptions,

- There are dependent relationships of jobs, and relationships can be denoted by directed acyclic graph (DAG) as in Figure 1.
- Jobs come in batch mode.
- Resources can not be occupied by other job when a job is running on them.

We model a workflow application as a DAG, let Γ be the finite set of tasks T_i $(1 \leq i \leq n)$. Let Λ be the set of directed arcs of the form (T_i, T_j) where T_i is called a parent task of T_j, and T_j the child task of T_i. Associated with each directed arc is a data flow in which the output of the parent is required as input data by the child. We assume that a child task cannot be executed until all of its parent tasks have been completed. Then, the workflow application can be described as a tuple $\Omega(\Gamma, \Lambda)$. Given a set of jobs $T = \{T_i\}, i = 1, 2, \ldots, n$ and a set of services $S = \{S_i\}, j = 1, 2, \ldots, m$, under the constraint that there are dependent relationships among jobs. The execution optimization problem is to generate a solution $I = (T_i, f_j(T_i))$ where $f_j(T_i)$ is a mapping function that assigns T_i onto service S_j. We also denote that $time(T_i, f_j(T_i))$ is the completion time of I and $cost(T_i, f_j(T_i))$ is the input data transmission cost and service cost for processing I. The goal is described as follows:

$$minimize \quad F = (F_1, F_2) \tag{3}$$

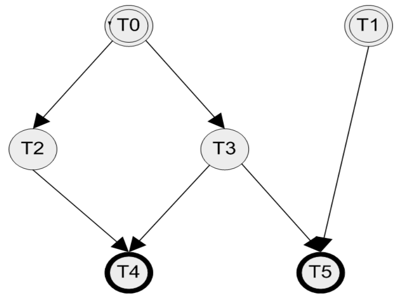

Figure 1: A Simple DAG

$$F_1 = \max_{T_i \epsilon \Gamma, f_j(T_i) \epsilon \Lambda} time(T_i, f_j(T_i)) \tag{4}$$

$$F_2 = \sum_{T_i \epsilon \Gamma, f_j(T_i) \epsilon \Lambda} cost(T_i, f_j(T_i)) \tag{5}$$

with subject to $F_1 < B$ and $F_2 < D$ where B is the cost constraint(budget) and D is he time constraint(deadline) required by users for workflow execution.

4. MULTIOBJECTIVE DIFFERENTIAL EVOLUTION

The idea of multiobjective differential evolution (MODE) was first introduced in [1]. In differential evolution (DE) [11], new candidate solutions are created by combining the parent individual and several other individuals of the same population. A candidate replaces the parent only if it has a better fitness value. This is a rather greedy selection scheme that often outperforms traditional Evolutionary Algorithms. Many variants of creation of a candidate are possible. We use the DE scheme DE/rand/1/bin described in Algorithm 1 (more details on this and other DE schemes can be found in [11]). When applying DE to MOPs, we face many difficulties. Besides preserving a uniformly spread front of nondominated solutions, which is a challenging task for any MOEA, we have to deal with another question, that is, when to replace the parent with the candidate solution. In single-objective optimization, the decision is easy: the candidate replaces the parent only when the candidate is better than the parent. In MOPs, on the other hand, the decision is not so straightforward. We could use the concept of dominance (the candidate replaces the parent only if it dominates it), but this would make the greedy selection scheme of DE even greedier. Therefore, MODE applies the following principle (see Algorithm 2). The candidate replaces the parent if it dominates it. If the parent dominates the candidate, the candidate is discarded. Otherwise (when the candidate and parent are nondominated with regard to each other), the candidate is added to the population. This step is repeated until *popSize* number of candidates are created. After that, we get a population of the size between *popSize* and 2·*popSize*. If the population grows, we have to truncate it to prepare it for the next step of the algorithm.

4.1 The Model

In order to extend MOEAs to solve the workflow scheduling problem, we need to define an appropriate problem representation, fitness assignment, and genetic operators. The methods we have employed are described in the following

Algorithm 1 CreateCandidate(Parent P_i<mat,ss>)

```
Pick three random individual $P_{i1}$,$P_{i2}$ and $P_{i3}$ where ($i1 \neq i2 \neq i3$)
Calculate Candidate C as, $C = P_{i1} + F \cdot (P_{i2} - P_{i3})$
Modify the Candidate by binary crossover with the Parent
return Candidate C
```

Algorithm 2 MODE

```
Define population $P$ with $popSize$ number of individual.
$P = \{P_1, P_2, \ldots, P_{popSize}\}$
Evaluate the initial population $P$ of random individuals.
while stopping criterion not met do
  for all $i$ such that $0 \leq i \leq popSize$ do
    $C$ = CreateCandidate($P_i$)
    Evaluate Candidate
    if Candidate $C$ dominates Parent $P_i$ then
      Candidate replaces the Parent
    else if Parent $P_i$ dominates Candidate $C$ then
      Parent replaces the Candidate
    else
      Add Candidate $C$ to the population
      $popSize \Leftarrow popSize + 1$
    end if
  end for
  If the population exceeds $popSize$, truncate it.
  Randomly enumerate individuals in $P$
end while
```

sub-sections. We have extended our ideas from [18] and [13].

4.1.1 *Encoding*

In this approach, each chromosome consists of two parts: the matching string and the scheduling string. Let *mat* be the matching string, which is a vector of length $|T|$, such that $mat(i) = j$, where $0 \leq i < |T|$ and $0 \leq j < |S|$; i.e., subtask T_i is assigned to service S_j. In terms of string representation it is T_i:S_j The scheduling string is a topological sort of the DAG, i.e., a total ordering of the nodes (subtasks) in the DAG that obeys the precedence constraints. Define *ss* to be the scheduling string, which is a vector of length $|T|$, such that $ss(k) = i$, where $0 \leq i, k < |T|$, and each T_i appears only once in the vector, i.e., subtask T_i is the k^{th} subtask in the scheduling string. Because it is a topological sort, if $ss(k)$ is a consumer of a global data item produced by $ss(j)$, then $j < k$. The scheduling string gives an ordering of the subtasks that is used by the evaluation step. Then in this approach, a chromosome is represented by a two-tuple <mat,ss>. Thus, a chromosome represents the subtask-to-service assignments (matching) and the execution ordering of the subtasks assigned to the same service. The scheduling of the *global data item transfers* and the *relative ordering of subtasks assigned to different services* are determined by the evaluation step. Figure 2 illustrates two different chromosomes for the DAG in Figure 1, for $|T| = 6$ and $|S| = 3$

4.1.2 *Initial Population*

In the initial population, a predefined number of solutions (chromosomes) are generated. When generating a chromosome, a new matching string *mat* is obtained by randomly

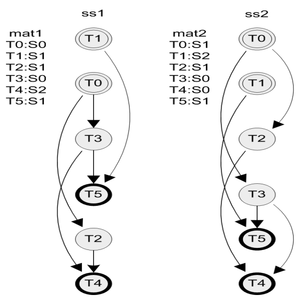

Figure 2: Two sample chromosomes from the DAG in Figure 1.

assigning each subtask to a machine. To form a scheduling string *ss*, the DAG is first topologically sorted to form a basis scheduling string. For each chromosome in the initial population, this basic string is mutated a random number of times (between one and the number of subtasks) using the mutation operator (described in Section 5.5) to generate the *ss* vector (which is a valid topological sort of the given DAG). Furthermore, it is common to incorporate solutions from some nonevolutionary heuristics into the initial population, which may reduce the time needed for finding a satisfactory solution. Since every time we are considering topological sort of the DAG, there is a nonzero probability that a chromosome can be generated to represent any possible solution to the matching and scheduling problem using the crossover and the mutation operators. (The crossover and the mutation operators will be discussed later sections).

4.1.3 *Fitness Measure*

A fitness function is used to measure the quality of the solutions according to the given optimization objectives. We separate fitness functions by objective functions and penalty functions. Objective functions are designed to encourage the algorithms to choose solutions with minimum objective values. The objective functions for solution I are defined as follows:

$$\text{Cost objective function } f_{cost}(I) = cost(I)/B \tag{6}$$

$$\text{Time objective function } f_{time}(I) = time(I)/D \tag{7}$$

A penalty function $P(I)$ is developed to handle constraints. It is defined as follows:

$$P(I) = P_{budget}(I) + P_{deadline}(I) \tag{8}$$

where P_{budget} is the budget penalty function defined by:

$$P_{budget} = \begin{cases} f_{cost}(I) & if\ cost(I) > B \\ 0 & otherwise \end{cases} \tag{9}$$

and $P_{deadline}$ is the deadline penalty function defined by:

$$P_{deadline} = \begin{cases} f_{time}(I) & if\ time(I) > D \\ 0 & otherwise \end{cases} \tag{10}$$

Since satisfying deadline and budget requirements is the primary goal of the scheduling scheme, the overall penalty is added to the objective functions to form the fitness functions:

$$\text{Cost fitness function: } F_{cost}(I) = f_{cost}(I) + P(I) \tag{11}$$

$$\text{Time fitness function: } F_{time}(I) = f_{time}(I) + P(I) \tag{12}$$

4.1.4 *Genetic Operators*

The crossover operator for the scheduling strings randomly chooses some pairs of the scheduling strings. For each pair, it randomly generates a cutoff point, which divides the scheduling strings of the pair into top and bottom parts. Then, the subtasks in each bottom part are reordered. In this way, in each crossover, we can create a valid schedule. Figure 3 demonstrates such a scheduling string crossover process. The crossover operator for the matching strings (See right side image of Figure 4) randomly chooses some pairs of the matching strings. For each pair, it randomly generates a cut-off point to divide both matching strings of the pair into two parts. Then the machine assignments of the bottom parts are exchanged. However, in our approach the crossover operator will be applied to both matching string and scheduling string $<mat,ss>$ but mutation will only be applied to scheduling string $<ss>$.The mutation operator works by randomly choosing a scheduling strings and it randomly selects a *victim* subtask. The valid range of the victim subtask is the set of the positions in the scheduling string at which this victim subtask can be placed without violating any data dependency constraints. Figure 5 shows an example of this mutation process. However, in the case of matching string *mat*, there will be no mutation, rather it will be subjected to the candidate creation of MODE (which will be discussed in the next sections). So our model only takes scheduling string for normal genetic operation but only the matching string will go through MODE operations.

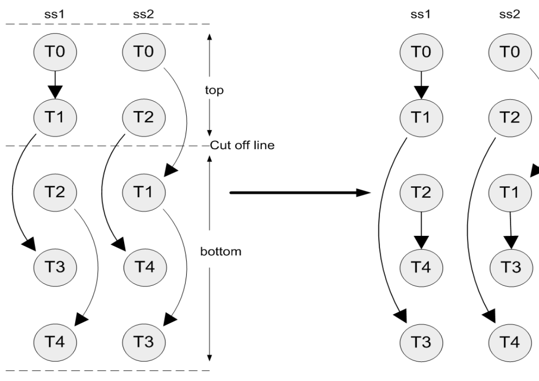

Figure 3: Scheduling string crossover

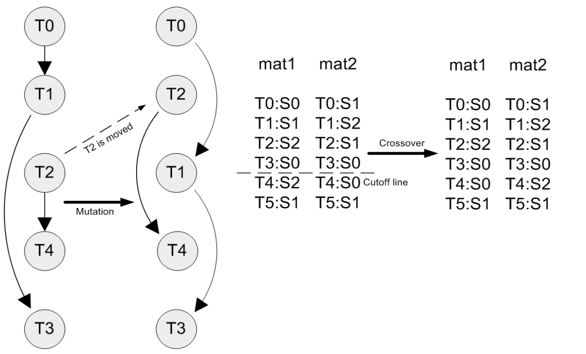

Figure 4: Scheduling String Mutation and Matching String Crossover

4.2 Creation of Candidate Solution

As described in section 4, the creation of candidate solution is done within step 2 of Algorithm 1 where candidate $C$is created as follows,

$$C = P_{i1} + F \cdot (P_{i2} - P_{i3}) \tag{13}$$

The matching string is an array of numbers or characters (each of the characters defines a service to be used by a specific task), it seems quite confusing to measure the quantity $P_{i2} - P_{i3}$ from two matching string of P_{i2} and P_{i3}. Let us suppose, the matching strings of P_{i2} and P_{i3} are $P_{i2}<mat> = \{1,2,3,4,5,6\}$ and $P_{i3}<mat> = \{2,5,3,1,4,6\}$ respectively. When we are going to compare two ordinal vectors, we will consider the first one as a 'reference' and second one as the 'target'. In [2],the 'reference'-vector is denoted as 'pattern-vector' and the 'target'-vector is denoted as 'disorder-vector'. Similarly, we considered $P_{i2}<mat> = \{1,2,3,4,5,6\}$ as pattern-vector and $P_{i3}<mat> = \{2,5,3,1,4,6\}$ as disorder-vector. Our goal is to measure the similarity between strings. In our experiment we have used the Ulam distance [2] to measure the quantity $(P_{i2} - P_{i3})$. Since it measures the disorder of ordinal variables by counting the minimum number of "Delete-Shift-Insert" operations. So in above case, the Ulam distance will be 2, since in $P_{i3}<mat>$, 1 should be moved to place between 2 and 5, and again 5 should be moved to place between 4 and 6 (Total 2 "Delete-Shift-Insert" operations). For simplicity we have chosen $F = 1$. However in the case of general DE, each of the variables are real numbers and the fitness are measured in terms of mathematical operations on those real numbers. But in our case, each of these variables (characters/numbers) refer to different service, so we cannot map these numbers/characters to real numbers and apply mathematical operations as in [8] and [9]. Consequently, in place of adding the difference with each value of P_{i1}, we mutate $P_{i1}<mat>$. i.e., we mutated D number of genes in P_{i1} where D is the Ulam distance measure described as above. The process is done in Algorithm 4, step 3. As result of this approach, we have just replaced the operation in equation 13 with Algorithm 3. We can now devise our algorithm from Algorithm 2 and 3. The modified MODE is illustrated in Algorithm 4.

Algorithm 3 ModifiedCreate(Parent $P_i<mat,ss>$)

```
Pick three random individual $P_{i1}$,$P_{i2}$ and $P_{i3}$ where $(i1 \neq i2 \neq i3)$
D = $UlamDistance(P_{i2}<mat>,P_{i3}<mat>)$
Candidate,$C<mat> \Leftarrow mutate(P_{i1}<mat>, D)$
Candidate,$C<ss> \Leftarrow Mutatate_{MatchingString}(P_{i1}<ss>)$
$C<ss> \Leftarrow Crossover_{SchedulingString}(C<ss>, P_i<ss>)$
$C<mat> \Leftarrow Crossover_{MatchingString}(C<mat>, P_i<mat>)$
return $C<mat,ss>$
```

5. EXPERIMENTS

In our experiment, we have implemented the balanced workflow described in [12, 15]. The workflows were designed to implement some specific parallel numerical computation problems such as Parallel Gauss-Jordan Algorithm to solve systems of equations [12], Parallel LU decompositions [12] and Discrete Laplace Transformation [15]. The example

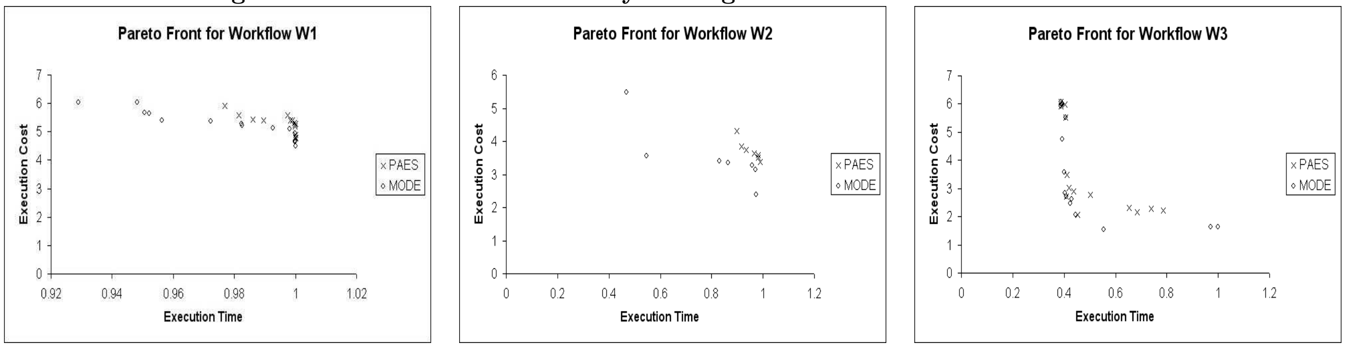

Figure 5: Pareto front obtained by two algorithms on different workflows

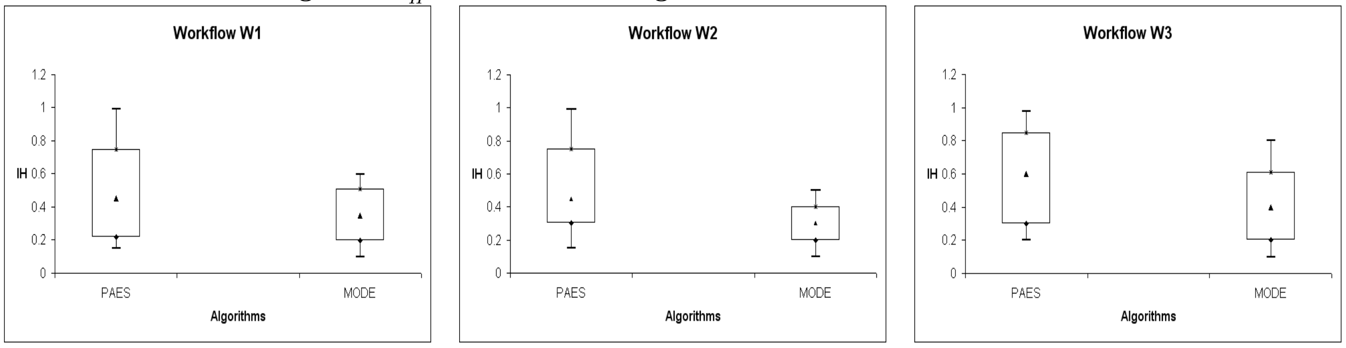

Figure 6: I_H^- measure of two algorithms on different workflows

Algorithm 4 ModifiedMODE

```
Define population P with $popSize$ number of individual.
$P = \{P_1{<}mat,ss{>}, P_2{<}mat,ss{>}, \ldots, P_{popSize}{<}mat,ss{>}$
Evaluate the initial population P of random individuals.
**while** stopping criterion not met **do**
  **for all** $i$ such that $0 \le i \le popSize$ **do**
    $C{<}mat,ss{>}$ = ModifiedCreate($P_i{<}mat,ss{>}$)
    Evaluate Candidate
    **if** Candidate $C{<}mat,ss{>}$ dominates Parent $P_i{<}mat,ss{>}$ **then**
      Candidate replaces the Parent
    **else if** Parent $P_i{<}mat,ss{>}$ dominates Candidate $C{<}mat,ss{>}$ **then**
      Parent replaces the Candidate
    **else**
      Add Candidate $C{<}mat,ss{>}$ to the population
      $popSize \Leftarrow popSize + 1$
    **end if**
  **end for**
  If the population exceeds $popSize$, truncate it.
  Randomly enumerate individuals in $P$
**end while**
```

Table 1: Selected Workflow Models

Workflow	#Nodes	Reference	Note
W1	15	[12]	Gauss-Jordan Algorithm
W2	14	[12]	LU decomposition
W3	16	[15]	Laplace Transform

workflow models are given in Table 1. The resources were 10 computing entities connected with each other with randomly assigned price levels. The parameter setting for the PAES (Pareto Archived Evolutionary Strategy) was taken from the original works [7] and our model also has the same settings as PAES. We have run each of the algorithms for 100 generations. The distance measure that we have used for our modified MODE was Ulam distance since it depends on the minimum number of "Delete-Shift-Insert" operations (as described in Section 6) and these operations best reflect the difference of resource arrangements in two schedule. In order to compare the performance of alternative workflow multi-objective scheduling algorithms, we need to examine the extent of minimization of the obtained non-dominated solutions produced by each algorithm for each objective and the spread of their solutions. Figure 5 shows the non-dominated solutions obtained at the end of simulation trial (average over 30 runs) for the workflow structure discussed in section 7. For all problems, the pareto optimal front obtained by our model is better than solutions found by PAES. In order to present a comprehensive comparison of the overall quality of these alternative approaches, we have run each algorithm for three different workflow structures [12],[15]. The experiment for each scenario was repeated 30 times (with different random seeds). We have constructed a reference set, R, by merging all of the archival non-dominated solutions found by each of the algorithms for a given workflow structure across 30 runs. We then use the hypervolume difference indicator I_H^- [6] to measure the differences between non-dominated fronts generated by the algorithms and the reference set R. I_H^- measures the portion of the objective space that is dominated by R. The lower the value of I_H^-, the better the algorithm performs. Box plots in

Figure 6 (I_H^- indicators for 3 different workflow examples) clearly prove that our approach is better than PAES.

6. CONCLUSION AND FUTURE WORK

In this paper,we have proposed a workflow execution planning approach, which optimize multiple objectives. The planner can generate a set of widespread alternative solutions if the optimization objectives are conflicted. Providing these alternative solutions can offer more flexibility to users to estimate their preferences and choose a desired workflow schedule based on their QoS requirements. Our MODE approach differs from others in the sense that we are dealing with real scheduling sequences rather than character to number transformation as adopted in[8] and [9]. We have compared our result with PAES and the I_H^- indicator shows that our approach performs better than PAES. Moreover our model is also free from extra computational overhead due to crowding distance sorting. In the case of candidate creation, we have only considered the Ulam distance and there is a lot of opportunities to investigate the algorithms performance with different string similarity metrics.

Multiobjective optimization in Grid scheduling is not a matured field. It still requires a number of detailed benchmark problems to test every type of multiobjective scenario on Grid scheduling or real life data to test an algorithm's performance. There are also a limited number of studies that have considered the flow-shop/job-shop scheduling problem using DE, however multiobjective scheduling poses additional challenges. Most of the widely used existing workflow scheduling algorithm only attempt to minimize either execution time or execution cost. However, additional objectives must be considered when scheduling workflows on utility Grids.

7. REFERENCES

[1] H. A. Abbass, R. Sarkar, and C. Newton. A Pareto Differential Evolution Approach to Vector Optimisation Problems. In *IEEE Congress on Evolutionary Computation*. IEEE Press, 2001.

[2] D. N. Bhat. An Evolutionary Measure for Image Matching. In *14th International Conference on Pattern Recognition*, pages 850–852. IEEE Press, August 1998.

[3] E. Deelman. Mapping Abstract Complex Workflows Onto Grid Environments. *Journal of Grid Computing*, 1:25–39.

[4] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco, California.

[5] X. He, X. H. Sun, and G. V. Laszewski. QoS Guided Min-Min Heuristic for Grid Task Scheduling. *Journal of Computer Science and Technology*, 18(4):442–451.

[6] S. Huband, P. Hingston, L. Barone, and L. While. A Review of Multiobjective Test Problems and a Scalable Test Problem Toolkit. *IEEE Transaction on Evolutionary Computation*, 10(5):477–506.

[7] J. Knowles and D. Corne. The Pareto Archived Evolution Strategy : A New Baseline Algorithm for Pareto Multiobjective Optimisation. In *The Congress on Evolutionary Computation*, pages 98–105. IEEE Press, 1999.

[8] A. C. Nearchou and S. L. Omirou. Differential Evolution for Sequencing and Scheduling Optimization. *Journal of Heuristics*, 12:395–411.

[9] G. Onwubolu and D. Davendra. Scheduling Flowshops using Differential Evolution Algorithm. *Europian Journal of Operational Research*, 171:674–692.

[10] R. Prodan and T. Fahringer. Dynamic Scheduling of Scientific Workflow Applications on The Grid: A Case Study. In *ACM symposium on Applied computing*. ACM Press, 2005.

[11] R. Storn and K. Price. Differential Evolution-A Simple and Efficient Heuristic for Global Optimization Over Continuous Spaces. *Journal of Global Optimization*, 11:241–354.

[12] T. Tsuchiya, T. Osada, and T. Kikuno. Genetic-Based Multiprocessor Scheduling using Task Duplication. *Microprocessors and Microsystems*, 22:197–207.

[13] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski. Task Matching and Scheduling in Heterogeneous Computing Environments using a Genetic-Algorithm-Based Approach. *Journal of Parallel Distributed Computing*, 47:9–22.

[14] A. S. Wu, H. Yu, S. Jin, K.-C. Lin, and G. Schiavone. An Incremental Genetic Algorithm Approach to Multiprocessor Scheduling. *IEEE Transaction on Parallel and Distributed Systems*, 15(9):824–834.

[15] M. Wu and D. Gajski. Hypertool: A Programming Aid for Message-Passing Systems. *IEEE Transaction on Parallel and Distributed Systems*, 1(3):330–343.

[16] G. Ye, R. Rao, and M. Li. A Multiobjective Resource Scheduling Approach Based on Genetic Algorithms in Grid Environment. In *5th International Conference on Grid and Cooperative Workshops*, 2006.

[17] J. Yu and R. Buyya. Scheduling Scientific Workflow Applications with Deadline and Budget Constraints using Genetic Algorithms. *Scientific Programming*, 14:217–230.

[18] J. Yu, M. Kirley, and R. Buyya. Multi-objective Planning for Workflow Execution on Grids. In *8th IEEE/ACM International Conference on Grid Computing*, September 2007.