

Mapping Heavy Communication Grid-Based Workflows onto Grid Resources Within An SLA Context Using Metaheuristics

Dang Minh Quan¹
D. Frank Hsu²

Abstract

Service Level Agreements (SLAs) is currently one of the major research topics in grid computing. Among many system components for the SLA-related grid jobs, the SLA mapping mechanism has received wide spread attention. It is responsible for assigning sub-jobs of a workflow to a variety of grid resources in a way that meets the user's deadline and costs as little as possible. With the distinguished workload and resource characteristics, mapping a heavy communication workflow within an SLA context gives rise to a complicated combinatorial optimization problem. This paper presents the application of various metaheuristics and suggests a possible approach to solve this problem. Performance measurements deliver evaluation results on the quality and efficiency of each method.

Key words: Grid computing, workflow, mapping, planning, Service Level Agreement

1. Introduction

Mapping and running jobs on suitable resources are the core tasks in grid computing. With the case of a grid-based workflow, where a single job is divided into several sub-jobs, the majority of efforts (Deelman et al, 2004; Spooner et al, 2003; Lovas et al, 2004) work to map a workflow onto the grid resources in the best effort manner. In the SLA context (Sahai, 2002), where resources are reserved to ensure the Quality of Service (QoS), mapping a workflow onto the grid resources requires a different mechanism. Some proposed solutions for this problem have appeared recently (McGough et al, 2005; Zeng et al, 2004; Brandic et al, 2004). However, most of the proposed mechanisms suppose a workflow including many sub-jobs, which are sequential programs, and a grid service having ability to handle only one sub-job at a time. This is not sufficient as sub-jobs in many existing workflows (Ludtke et al, 1999; Berriman et al, 2003; Richter et al, 2004) are parallel programs and many high performance computing centers (HPCC) provide computing service under single grid service (Burchard et al, 2004). It is obvious that one HPCC can handle many sub-jobs, either sequential programs or parallel programs, at a time. Moreover, none of them considered the case of having heavy communication among sub-jobs in the workflow. This paper, which is a continuous work in a series of efforts supporting SLA for the grid-based workflow (Quan and Kao, 2005abcd; Quan, 2006; Quan and Hsu, 2006), presents the application of various standard metaheuristics and suggests a possible approach to solving this problem.

1.1 Workflow Model

Like many popular systems handling grid-based workflow (Deelman et al, 2004; Spooner et al, 2003; Lovas et al, 2004), our proposed workflow system also uses the Directed Acyclic Graph (DAG) form. The user specifies the required resources needed to run a sub-job, data transfer among sub-jobs, and the estimated runtime of sub-jobs. The users also impose the expected runtime of the whole workflow. The user wants the system to finish running the whole workflow in time. In the scope of this paper, the time is computed in slots. Each slot equals a specific period of real time. Figure 1 presents a concrete example of a grid workflow. Each sub-job of the workflow has different resource requirements as in Table 1.

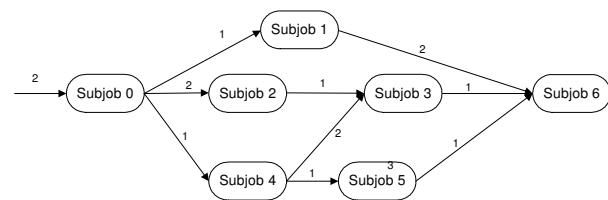


Fig. 1: A sample grid-based workflow

¹ School of Information Technology, International University in Germany, 76646 Bruchsal, Germany; quandm@upb.de

² Department of Computer & Information Sciences, Fordham University, 113 West 60th Street New York, NY USA; hsu@cis.fordham.edu

It is noted that a sub-job of the workflow can be either a sequential program or a parallel program and that the data to be transferred among sub-jobs is very large, usually on the GB scale. The case of light communication among sub-jobs of the workflow was handled in (Quan, 2005d).

Table 1: Resource requirements for sub-jobs

Sj_ID	CPU	Storage	Exp	Runtime
0	18	59	1	6
1	16	130	3	8
2	20	142	4	5
3	22	113	4	8
4	18	174	2	9
5	20	97	3	14
6	16	118	1	4

1.2 Grid-based Service Model

A computational grid includes many high performance computing centers (HPCCs). We believe that only HPCCs have enough resources to support SLA for sub-jobs of a workflow. The resources in each HPCC are managed by the software called local Resource Management System (RMS). In this paper, the acronym ‘‘RMS’’ is used to represent an HPCC as well as the grid service of that HPCC. Each RMS has its own unique resource configuration. Those differences include the number of CPUs, amount of memory, storage capacity, software systems, experts and service prices. To ensure that a sub-job can be executed within a dedicated time period, the RMS must support advance resource reservations such as CCS (Hovestadt, 2003). Figure 2 depicts a sample CPU reservation profile in a typical RMS. Queuing-based RMSs are not suitable for our requirement, as no information about the starting time is provided. In our system, we reserve three main types of resources: CPUs, storages and experts. An extension to other devices, software licenses, and other related resources is straightforward.

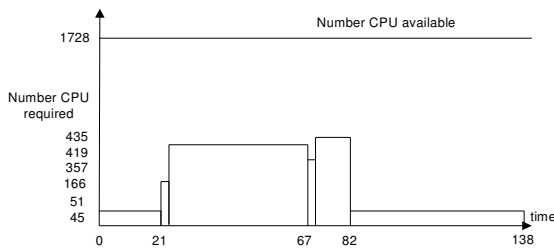


Fig. 2: A sample CPU reservation profile of a local RMS

If two subsequent sub-jobs are executed by the same RMS, it is not necessary to do data transfer and the time used for this work equals zero. Otherwise, data transfer tasks between the RMS’s must be performed. To make sure that a specific amount of data will be transferred within a specific period of time, a bandwidth must also be reserved.

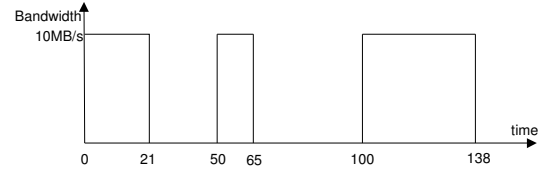


Fig. 3: A sample bandwidth reservation profile of a link between two local RMSs

Up until now, we are not aware of any mechanism responsible for the above mentioned task in the worldwide network. In order to overcome that disadvantage, we use a central broker mechanism. The link bandwidth between two local RMSs is determined as the average bandwidth between two sites in the network, which has a different value with each different RMS couple. Whenever a data transfer task is on a link, the SLA broker will determine which time slot is available for that task. During that specified period, the task can use the whole bandwidth and other tasks must wait. Using this principle, the bandwidth reservation profile of a link will look similar to the one depicted in Figure 3. A more accurate model with bandwidth estimation (Wolski, 2003) can be used to determine the bandwidth within a specific time period instead of the average value. In both cases, the main mechanism is unchanged.

1.3 Mapping Mechanism Requirement

A formal specification of the described problem entails the following elements:

1. Let R be the set of all grid RMSs. It includes a finite number of RMSs, which provide static information about controlled resources and the current reservations/assignments.
2. Let S be the set of all sub-jobs in a given workflow. It includes all sub-jobs with the current resource and deadline requirements.
3. Let E be the set of all data transfers in a workflow. It indicates the dependency between the sub-jobs and the necessity for data transfers between the sub-jobs.
4. Let K_i be the set of resource candidates for sub-job s_i . It includes all RMSs that can run sub-job s_i . We note that K_i is a subset of R .

Based on the given input, we are looking for a feasible and possibly optimal solution, which allows the most efficient mapping

of the workflow in a grid environment with respect to the given global deadline. The required solution is a set defined in Formula 1.

$$M = \{(s_i, r_j, start_slot) \mid s_i \in S, r_j \in K_i\} \quad (1)$$

If a solution does not have *start_slot* for each s_i , it becomes a configuration defined in Formula 2.

$$a = \{(s_i, r_j) \mid s_i \in S, r_j \in K_i\} \quad (2)$$

A feasible solution must satisfy the following conditions:

Criterion 1: All $K_i \neq \emptyset$. There is at least one RMS in the candidate set of each sub-job.

Criterion 2: The total runtime period of the workflow must be within the expected period given by the user.

Criterion 3: The dependency among the sub-jobs is resolved and the execution order remains unchanged.

Criterion 4: Each RMS provides a profile of currently available resources and can run many sub-jobs of a single flow both sequentially and in parallel. Those sub-jobs, which run on the same RMS, form a profile of resource requirement. With each RMS r_j running sub-jobs of the grid workflow having each time slot in the profile of available resources and resource requirements, the number of available resources must be larger than that of the resource requirements.

Criterion 5: The data transmission task e_{ki} from sub-job s_k to sub-job s_i must not overlap with any other reserved data transmission task on the link between the RMS running sub-job s_k and the RMS running sub-job s_i , $e_{ki} \in E$.

In the next phase, feasible solutions with the lowest cost are sought. The cost C of a grid workflow is defined in formula 3. It is a sum of four factors: the cost of using: (1) the CPU, (2) storage, (3) expert knowledge, and (4) data transfer between resources.

$$C = \sum e_{ki} \cdot n_d * r_j \cdot p_d + \sum_{i=1}^n s_i \cdot r_i * (s_i \cdot n_c * r_j \cdot p_c + s_i \cdot n_s * r_j \cdot p_s + s_i \cdot n_e * r_j \cdot p_e) \quad (3)$$

where $s_i \cdot r_i$, $s_i \cdot n_c$, $s_i \cdot n_s$, $s_i \cdot n_e$ are the runtime, number of CPU, number of storage, and number of experts of sub-job s_i respectively. $r_j \cdot p_c$, $r_j \cdot p_s$, $r_j \cdot p_e$ and $r_j \cdot p_d$ are the price of using CPU, storage, expert, and data transmission of RMS r_j respectively.

$e_{ki} \cdot n_d$ is the amount of data to be transferred from sub-job s_k to sub-job s_i .

If two sequential sub-jobs run on the same RMS, the cost of transferring data from the previous sub-job to the latter sub-job is neglected. As the data to be transferred among sub-jobs in the workflow are huge, it is necessary to do bandwidth reservation in order to ensure the deadline of the workflow. In this case, the time to do the data transmission task becomes unpredictable. It depends on the bandwidth and the reservation profile of the link, which varies from link to link. The variety in completion time of the data transmission has a great impact on the total runtime of the workflow. It can be shown easily that the optimal mapping of a workflow onto a grid-based RMS with optimized cost is a NP-hard problem.

2. Related Works

The algorithm that maps each sub-job separately on an individual grid RMS is presented in (Cao et al, 2005). The algorithm processes one sub-job at a time and schedules it to a suitable RMS where start time slot does not conflict with the dependency of the flow. The selection of the destination of resources is optimized with respect to a minimal completion time. When applying this strategy to a specified problem, each sub-job will be assigned separately to the cheapest feasible RMS. This strategy allows fast computation of a feasible schedule, but it does not consider the entire workflow. Neither does it take into consideration the dependencies among the inter-dependency among various sub-jobs.

The mapping of grid workflows onto grid resources based on existing planning technology is presented in (Deelman et al, 2004). This work focuses on coding the problem to be compatible with the input format of specific planning systems. It transfers the mapping problem to a planning problem. Although this is a flexible way to achieve different goals, significant disadvantages regarding the time-intensive computation, long response times and the missing consideration of grid-specific constraints exist. The later character is the main cause that the suggested solutions often do not meet the expected quality requirement.

In (Chen and Yang, 2006ab), a checkpoint selection strategy, named CSSMTR (Minimum Time Redundancy based Checkpoint Selection Strategy) is presented. Checkpoint selection strategies are used to select checkpoints for verifying fixed-time constraints at run-time execution stage. The checkpoints selected by CSSMTR dynamically during grid workflow execution are not only necessary but also sufficient features to help reduce the cost of executing workflow. Hence, the unnecessary and omitted temporal verification can be avoided. It can also improve the overall temporal verification efficiency and effectiveness significantly. Our approach differs from the work in (Chen and Yang, 2006ab) in that we consider the QoS aspect and the cost of the overall workflow at the beginning of the planning phase.

Zeng et al (2004) and Brandic et al (2004) built systems to support QoS features for grid-based workflow. In their work, a workflow includes many sub-jobs which are sequential programs. Also, a grid service has the ability to handle one sub-job at a time. To map the workflow onto the grid services, they used the Integer Programming method. Applying the Integer Programming to our problem leads to many difficulties. The first is the flexibility in computing the runtime of a data transfer task. The time to complete data transfer task depends on the bandwidth and the reservation profile of the link, which varies from link to link. The variety in completion time of data transfer task makes the presentation of constraints very complicated. The second is that a RMS can handle many parallel programs at a time. Thus, presenting the constraints between the profile of resource requirements and that of resources available in Integer Programming has become quite difficult.

Our problem has a close relation to the classical job shop scheduling problem (JSSP) (Brucker, 2004). The DAG form of our workflow is similar to the graph representing the sequence processing of JSSP. Each sub-job in our problem is correlative to each operation in JSSP. Since a job shop scheduling problem is an NP hard problem, two main methods to solve this problem, **complete** and **incomplete method**, exist. A complete method explores systematically, though very often implicitly, the whole search space. To do this, most of complete methods construct, in a "step by step" way, a solution and backtrack in case of failure (Kumar, 1992). These methods usually use various heuristics (Sadeh and Fox, 1996; Kumar, 1992; Gent et al, 1996) to guide the choice of the next variable to be instantiated and how its values are assigned. They also employ powerful filtering techniques to achieve different levels of consistency. Similarly, exact methods for constraint optimization are usually based on the branch and bound principle (Kumar, 1992) and they try to eliminate heuristically as many solutions as possible which do not lead to an optimum. Complete and exact methods have in general exponential time complexity. The solution time required by such a method may consequently become prohibitive for large size problems.

An incomplete (not-exact) method does not systematically explore the whole search space. Instead, it tries to examine as rapidly as possible a large number of search points according to a selective or random strategy. Local search is one of the most popular examples of this family of methods. In general, these methods do not guarantee the completeness of the resolution but require no exponential time complexity. They constitute a very interesting alternative for practical solution of many hard and large size problems. Popular methods in this direction include Tabu Search (Glover, 1989,1990), Simulated Annealing (Kirkpatrick et al, 1983; Dowsland, 1993), and GA (Kacem et al, 2001,2002).

In the literature, when applying local search to the problem, the moving neighborhood is the most important factor in determining the speed of the algorithm and the quality of the solution. Many

effective neighborhood structures such as N1, N2, N3, N4 (Brucker45, 2001) and N5 (Nowicki and Smutnicki, 1996,2005) were proposed. The main activity of these neighborhood structures is changing the process sequence of two operations in the same machine. However, since each operation in the JSSP can be mapped to only one machine and one machine can process only one operation at a time, each sub-job in our problem can be mapped to several RMSs and each RMS can process several sub-jobs at a time. Therefore, such process of sequence changing does not exist in our solution.

The flexible job shop scheduling problem (FJSSP) extends the JSSP by assuming that, for each given operation, there exist several instances of the machine type necessary to perform the job. The method of FJSSP is far more complicated than the classical JSSP with two main problems: (a) assigning each operation to an appropriate machine, and (b) sequencing the operation in each machine. To solve the FJSSP, researchers have applied several local search methods and proposed several techniques (Mastrolilli and Gambardella, 2000; Jansen et al, 2000) to reduce the large search space to a smaller one. Those techniques are based strictly on each machine doing one operation at a time. Our problem differs from the FJSSP approach in that while each machine in FJSSP can process only one operation at a time, each RMS in our problem can process several sub-jobs at a time. Thus, applying the FJSSP proposed techniques to our problem is difficult.

Related to our problem, there is a multiprocessor scheduling precedence-constrained task graph problem (Gary and Johnson, 1979; Kohler and Steiglitz, 1974). Since this is a well-known problem, the literature recorded many solutions, which can be classified into several groups (Kwok and Ahmad, 1999). One classical approach is based on the so-called list scheduling technique (Adam et al, 1974; Coffman, 1976). Other approaches are Unbounded Number of Cluster (UNC) Scheduling (Gerasoulis and Yang, 1992; Sarkar, 1989), Bounded Number of Processors (BNP) Scheduling (Adam et al, 1974; Kruatrachue and Lewis, 1987), Task Duplication Based (TDB) Scheduling (Colin and Chretienne, 1991; Kruatrachue and Lewis, 1987), Arbitrary Processor Network (APN) Scheduling (Rewini and Lewis, 1990; Sih and Lee, 1993), and genetic algorithms (Shahid et al, 1994; Hou et al, 1994). Our problem differs from the multiprocessor scheduling precedence-constrained task graph problem in many factors. In the multiprocessor scheduling problem, all processors are similar. But in our problem, RMSs are heterogeneous. Each task in our problem can be a parallel program, while each task in those other problems is strictly a sequential program. Each node in those problems can process one task at a time while each RMS in our problem can process several sub-jobs at a time. For these reasons, applying local search approach to those problems is very rare. In our solution, local search is employed and proved to be quite effective.

3. Mapping with Standard Metaheuristics

Our problem is a specific case of a combinatorial optimization (CO) problem. Metaheuristics are usually used to solve CO problems especially in mapping and scheduling. Therefore, our first step is to directly apply many popular metaheuristics to our problem to see the advantage and disadvantage of each method. The selected metaheuristics include Tabu Search (Glover, 1989,1990), Simulated Annealing (Kirkpatrick et al, 1983; Dowsland, 1993), Iterated Local Search (Stuetzle, 1999ab), Guided Local Search (Voudouris, 1997; Voudouris and Tsang, 1999), Genetic Algorithm (Kacem et al, 2001,2002), and Estimation of Distribution Algorithm (Pelikan et al, 1999ab). Other methods such as Variable Neighborhood Search (Hansen and Mladenovic, 2001), and Ant Colony Optimization (Dorigo and Di Caro, 1999) face many difficulties when applied to our problem. For example, Ant Colony Optimization is shown to be effective specifically for Traveling Salesman problem. Hence, converting to our case requires major modifications, which can decrease the performance of the original algorithm.

3.1 General Strategy

The input of the mapping procedure includes information about workflow and RMSs. The user provides necessary information of a workflow through the SLA text, which is described in (Quan and Kao, 2005ab). A parser extracts information about workflow and stores it to a file describing sub-jobs and to another file describing the dependence. The former contains various sub-jobs' resource requirements and estimated runtime. The latter contains the source sub-job, destination sub-job, and number of data transfers. Information about RMSs is stored in a relational database, which is specifically designed for the system. They include the description of the resource configuration in each RMS, resource reservation profile of each RMS, and bandwidth reservation profile of each link. Data about RMSs is collected by the monitoring module. Based on this information, the system then performs the mapping task.

```
1. Determine candidate RMSs for each sub-job.
   If resource in the Grid free {
2. Call an algorithm to find optimal cost solution
   }else {
3. Call an algorithm to find optimal runtime solution
   then call an algorithm to find optimal cost solution
   }
```

Fig. 4: Mapping mechanism overview

Figure 4 presents the basic strategy of a mapping mechanism. Each sub-job has different resource requirements about the type of RMS and the type of CPU. There are many RMSs with different resource configurations. The first action is finding the suitable

RMSs among those heterogeneous ones which can meet the requirement of the sub-job. Data of resource parameters of each RMS is represented by a number value and is stored in a separate column in the database table. For example, with the parameter "Operating System", OSs such as Linux, Sun, Window, and Unix are represented by numerical numbers 1, 2, 3, and 4, respectively. The correlative resource requirement parameter of the sub-job is also represented by a numerical value in the same manner. Thus, the matching between the sub-job's resource requirement and RMS's resource configuration is done by several logic checking conditions in the WHERE clause of the SQL SELECT command. This work satisfies Criterion 1. With the case of our example, we assume that each sub-job of the workflow depicted in Figure 1 has three RMS candidates R1, R2 and R3.

The second and the third steps require two different algorithms. The first is used to find a cost optimal solution, while the second is used to find a runtime optimal solution. In this paper, we concentrate on the cost optimal problem. An algorithm handling the runtime optimal problem is described in (Quan, 2006). The following sections describe a variety of different metaheuristics in the process of constructing various algorithms.

3.2 Tabu Search (TS)

TS (Glover, 1989,1990) is among the most cited and used metaheuristics for CO problems. The standard Tabu search uses a tabu list that keeps track of the most recently visited solutions and the process is prohibited from moving toward them. This allows Tabu Search both to escape from the local minima and to implement an explorative strategy. The TS algorithm to find the minimal cost of a workflow within an SLA context is presented in Figure 5.

```
Repeat {
  Pick randomly an initial configuration a
  Compute makespan p of a
} until p meets Criteria 2
Compute cost c of a
a*=a // a* is the final configuration, c* is the cost of a*
while(num_mv<max) {
  for each solution in the neighborhood set of a {
    compute the makespan and cost
    if makespan meets Criteria 2
      store tuple (sub-job, RMS, cost) to a list
  }
  Sort the list according to cost criteria
  Pick the best solution, which has cost<c* or not affect Tabu rule
  Assign tabu_number to the selected RMS
  if the selected solution has cost<c* then store in a*
  Store the selected solution in a
  num_loop++
}
```

Fig. 5: TS algorithm to find the minimal cost

Beside the standard components of Tabu Search, there are some components specific to the workflow problems.

The neighborhood set structure: A configuration can also be presented as a vector. The index of the vector represents the sub-job, and value of the element represents the RMS. With a configuration $a = a_1 a_2 \dots a_n$, $a_i \in K_i$, we generate $n*(m-1)$ configurations a' as in Figure 6. We change the value of x_i to every value in the candidate list which is different from the present value. With each change, we have a new configuration. After that we have set A , $|A|=n*(m-1)$. A is the set of neighborhoods of a configuration. A detailed neighborhood set for the case of our example is presented in Figure 7.

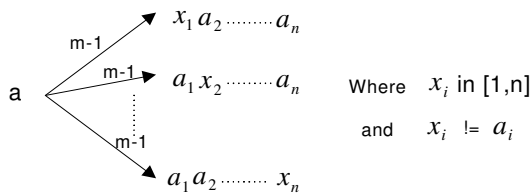


Fig. 6: Neighborhood structure of a configuration

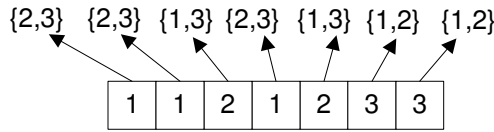


Fig. 7: Sample neighborhood structure of a configuration

The assigning sequence of the workflow: When the RMS executes each sub-job, the bandwidth among sub-jobs is determined. Next task is to determine a time slot to run the sub-job in the specified RMS. At this point, the assigning sequence of the workflow becomes important. The sequence of determining runtime for sub-jobs of the workflow in RMS can also affect the final makespan, especially in the case of having many sub-jobs in the same RMS.

In general, to ensure the integrity of the workflow, sub-jobs in the workflow are assigned according to the sequence of the data processing. However, this principle does not cover the case of a set of sub-jobs, which have the same priority in data sequence and do not depend on each other. To examine the problem, we determine the earliest and latest start times of each sub-job of the workflow in ideal condition. The time period to do data transfer among sub-jobs is computed by dividing the amount of data to a fixed bandwidth. The earliest and latest start, stop times for each sub-job, and data transfer depend only on the workflow topology and the runtime of sub-jobs, not on the resources context. These parameters can be determined using conventional graph algorithms. A sample of

these data for the workflow in Figure 1, in which the number above each link represents number of time slots to do data transfer, is presented in Table 2.

Table 2: Valid start time for sub-jobs of workflow in Figure 1

Sj_ID	Earliest start	Latest start
0	0	0
1	28	28
2	78	78
3	28	58
4	30	71
5	23	49
6	94	94

The ability to find a suitable resource slot for running a sub-job depends on the number of free resources during the valid running period. From the graph in Figure 1, we can see that sub-job 1 and sub-job 3 have the same priority in their data sequence. However, from the data in Table 2, sub-job 1 can start at max time slot 28 while sub-job 3 can start at max time slot 58 without affecting the finished time of workflow. Suppose that two sub-jobs are mapped to run in the same RMS and the RMS can run one sub-job at a time. If sub-job 3 is assigned first and in the worst case at time slot 58, sub-job 1 will be run from time slot 92. Thus the workflow will be delayed at least 64 time slots. If sub-job 1 is assigned first at time slot 28, sub-job 3 can be run at time slot 73 and the workflow will be late 15 time slots. Here we can see, the latest time factor is the main parameter to evaluate the full affection of the sequential assigning decision. It can be seen clearly that mapping a sub-job with the smaller latest start time first will make the latency smaller. Thus, the latest start time value determined above can be used to determine the assigning sequence. The sub-job having the smaller latest start time will be assigned earlier. This procedure will satisfies Criterion 3.

```

For each sub-job k following the assign sequence {
  For each link from determined sub-jobs to k{
    min_st_tran=end_time of source sub-job
    search reservation profile of the link to find
    start_tran > min_st_tran
    end_tran = start_tran+num_data/bandwidth
    store end_tran in a list
  }
  min_st_sj=max(end_tran)
  search in reservation profile of RMS running
  k the start_job > min_st_sj
  end_job= start_job + runtime
}

```

Fig. 8: Procedure to determine the timetable for the workflow

Determining the timetable of the workflow: To determine the makespan of the present solution, we have to determine the specific runtime period for each sub-job and each data transfer task. The start time of a data transfer task depends on the finish time of the source sub-job and the state of the link's reservation profile. We use *min_st_tran* variable to present the dependence on the finish time of the source sub-job. The start time of a sub-job depends on the latest finish time of the related data transfer tasks and on the state of the RMS's reservation profile. We use *min_sj_tran* variable to present the dependency on the latest finish time of the related data transfer tasks. The task to determine the timetable for the workflow is done with the procedure in Figure 8.

For each sub-job of the workflow in the assigning sequence, we first find all the runtime periods of data transfer tasks from previous sub-jobs to the current one. This time period must occur later than the finish time of the source sub-job. Note that with each different link, the transfer time is different because of different bandwidth. We then determine the runtime period of the sub-job itself. This time period must occur later than the latest finish time of any previous related data transfer task. Although the procedure is not complicated, it is time consuming. The most time consuming steps are in the process of searching reservation profiles. This procedure satisfies the Criteria 4 and 5.

Tabu Search: Starting from initial solution, the algorithm checks for the best configuration in the neighborhood of the current configuration at each iteration. The best neighbor that does not violate the Tabu rule is sought to replace the current configuration even if it is no better than the current configuration in terms of cost. If the best neighbor costs less than those of all the configurations found so far, it replaces the current configuration even if it violates the Tabu rule.

3.3 Simulated Annealing (SA)

SA is an advanced local search method that derives its inspiration from the physical annealing process studied in statistical mechanics (Kirkpatrick et al, 1983; Dowsland, 1993). An SA algorithm repeats an iterative repairing procedure, which looks for better solutions while offering the possibility of accepting some worse solutions in a controlled manner. The second feature allows SA to escape from the local optima. The SA algorithm to find the minimal cost of a workflow within an SLA context is presented in Figure 9. The neighborhood structure, the assigning sequence, and the procedure to determine the timetable are the same as those described in the Tabu Search section.

Temperature t , step length ℓ , and A are the parameters of an SA algorithm. Increasing t leads to increasing the possibility of accepting a worse solution. Decreasing ℓ makes the number of iterations for a move decrease. After several studies, we found suitable initial values of $t=2.5$; $A=100$; and $\ell =100$.

```
Repeat {
  Pick randomly an initial configuration a
  Compute makespan p of a
} until p meets Criteria 2
Compute cost c of a
a'=a // a' is the final configuration, c' is the cost of a'
Assign initial temperature t
Assign initial step length l
while(num_mv<max) {
  while(l_count<l) {
    Repeat {
      Pick randomly a configuration a' from the neighborhood of a
      Compute makespan p' of a'
    } until p' meets Criteria 2
    Compute cost c' of a'
    if((c'-c<0)or((c'-c>0) and (Probability exp(-(c'-c)/t) is verified)){
      a=a'
      if(c'<c")
        a'=a'
      num_mv++
      l_count++; n_ite++
    }
    t=(1-A/n_ite)
    l=(1-A/n_ite)
  }
}
```

Fig. 9: SA algorithm to find the minimal cost

Starting from the initial solution, the algorithm checks for a better configuration in the neighborhood of the current configuration of each iteration. When the number of iterations increases, we have: (a) the temperature increases, (b) the possibility to accept bad configurations increases, and (c) the number of checking a new configuration within a temperature decreases.

3.4 Iterated Local Search (ILS)

ILS is a simple but powerful metaheuristic algorithm (Stuetzle, 1999ab). It applies local search techniques to an initial solution until it finds a local optimum. Then, it perturbs the solution and restarts the local search. If the perturbation is too small, the algorithm might not escape from the local area. If the perturbation is too strong, the algorithm is similar to a random restart local search. The ILS algorithm to find the minimal cost of a workflow within an SLA context is presented in Figure 10. The neighborhood structure, assigning sequence, and procedure to determine the timetable are the same as those described in the Tabu Search section.

Local search algorithm is used to find the configurations which are local optimum in cost. The basic steps of this algorithm include:

1. Starting from the current configuration, we compute the cost of each of the configurations in the set of neighborhood configurations to choose the best one.

2. If the best found configuration has a smaller cost than the current configuration, we replace the current configuration with the new one and repeat the process.
3. Otherwise the search process stops.

```

Repeat {
  Pick randomly an initial configuration a
  Compute makespan p of a
} until p meets Criteria 2
a<- LocalSearch(a)
Compute cost c of a
a"=a // a" is the final configuration, c" is the cost of a"
m_loop=0 // m_loop stores times the configuration not improve
while(num_mv<max) {
  Repeat {
    a' <- Perturbation(a)
    Compute makespan p' of a'
  } until p' meets Criteria 2
  a'<- LocalSearch(a')
  Compute cost c' of a'
  if(c'<c){
    a=a'
    m_loop=0
    if(c'<c"){
      a"=a'; c"=c' }
  }
  else{
    m_loop++
    if(m_loop<max_val)
      a=a'
  }
}

```

Fig. 10: ILS algorithm to find the minimal cost

The perturbation procedure creates a new configuration from the current configuration by changing the assigned RMS of each of the sub-jobs. The candidate sub-job is selected randomly and the new RMS is chosen randomly in the candidate set. After a number of studies, we conclude that changing numbers approximately four times gives rise to much better and more accurate solutions. Thus, we use this value for every case in the experiment.

The ILS algorithm starts with an initial configuration. It performs perturbations to create the new ones and to execute local searches to improve it as far as possible. If the cost of the new configuration is smaller than the current one, it replaces the older one. If we cannot find any better configuration after a number of perturbations, the current configuration is also replaced with a new one that is randomly chosen.

3.5 Guided Local Search (GLS)

The basic GLS principle (Voudouris, 1997; Voudouris and Tsang, 1999) is to guide the search in the way that it would gradually move away from local minima by changing the search landscape. In GLS, the set of solutions and the neighborhood structure is kept fixed, while the objective function f is dynamically changed with

the aim of making the current local optimum "less desirable". The GLS algorithm to find minimal cost of a workflow within an SLA context is presented in Figure 11. The neighborhood structure, assigning sequence, and procedure to determine timetable are the same as those described in the Tabu Search section. The local search module used in the GLS algorithm is similar to the one in ILS. The objective function $f(a)$ is determined as in Formula 4.

```

Repeat {
  Pick randomly an initial configuration a
  Compute makespan p of a
} until p meets Criteria 2
Compute cost c of a
Initialize penalty kij=0 // each candidate RMS rj of sub-job i has a kij
a"=a // a" is the final configuration, c" is the cost of a"
while(num_mv<max) {
  Compute cost c of a
  f'=c + f(a)
  LocalSearch(a) finds local optima a according to f' function and
  records local optima a' according to c
  Compute cost c' of a'
  if(c'<c"){
    a"=a'
    c"=c' }
  Compute Util(a,rj,kij)
  kij++ if Util(a,rj,kij) max
}

```

Fig. 11: GLS algorithm to find the minimal cost

$$f(a) = \lambda * \sum (k_{ij} * I_{ij}(a)), \quad (4)$$

where a is the candidate configuration, λ is a parameter of the GLS algorithm. After a number of studies, we conclude that $\lambda=20$ gives rise to better results. Therefore, we used this value for all experiments. k_{ij} is the penalty of candidate RMS r_j of sub-job s_i . $I_{ij}(a)$ is an indication of whether $s_i:r_j$ exists in a . $I_{ij}(a)=1$ if $s_i:r_j$ exists in a ; and 0 otherwise. Function $Util(a,r_j,k_{ij})$ is determined in Formula 5.

$$Util(a, r_j, k_{ij}) = I_{ij}(a) * \frac{1}{1 + k_{ij}}. \quad (5)$$

The GLS algorithm starts with an initial configuration. It calls the local search to improve the configuration as far as possible according to the modified cost function. The higher frequency the r_j of s_i is selected before, the less it becomes attractive for the next selection. In such a situation, the other candidate RMS has a better chance to be selected. During the search process, the original cost of the candidate configuration is also computed and the best one is recorded.

3.6 Genetic Algorithm (GA)

GA (Kacem et al, 2001,2002) is a part of Evolutionary Computing techniques, which are inspired by Darwin's theory of natural selection. GA begins with a set of solutions called "population". Solutions from one population are selected according to their fitness and used to form a new population. This is motivated by a hypothesis that the new population may be better than the old one. This process is repeated a number of times to find out the best solution. The GA algorithm to find minimal cost of a workflow within an SLA context is presented in Figure 12.

```

Repeat{
  Repeat {
    Pick randomly an initial configuration a
    Compute makespan p of a
  } until p meets Criteria 2
  put a to the population set
}until the population includes n configurations
while(num_mv<max) {
  Evaluate the cost of each configuration
  a*= best configuration
  Add a" to the new population
  while the new population is not enough {
    Repeat{
      Select two parent configuration according to their makespan
      Crossover the parent with a probability to form new configuration
      Mutate the new configuration with a probability
      Compute makespan p of the new configuration
    } until p meets Criteria 2
    Put the new configuration to the new population }
  }
return a"

```

Fig. 12: GA algorithm to find the minimal cost

Parents are selected according to the roulette wheel method. The fitness of each configuration = 1/cost. First, the sum L of all configuration fitness is calculated. Then, a random number l from the interval (0,L) is generated. Finally, we go through the population to sum the fitness p . When p is greater than l , we stop and return the configuration to where we were in the last step.

The crossover point is chosen randomly. The child is formed by copying from two parts of the configuration. The mutation point is also chosen randomly. At each mutation point, r_j of s_i is replaced by another RMS in the candidate set. It is noted that the probability of having a child with mutation is rather low, ranging approximately from 0.5% to 1%.

3.7 Estimation of Distribution Algorithm (EDA)

EDA (Pelikan et al, 1999ab) is a new area of Evolutionary Computation. In EDAs, there is neither crossover nor mutation operators. New populations are generated by sampling the probability distribution, which is estimated from a database containing selected individuals of the previous generation. The

EDA algorithm to find the minimal cost of a workflow within an SLA context is presented in Figure 13.

```

Repeat{
  Repeat {
    Pick randomly an initial configuration a
    Compute makespan p of a
  } until p meets Criteria 2
  put a to the population set
}until the population includes n configurations
Call LocalSearch() to improve the population
while(num_mv<max) {
  Select the best half of the population
  a"=the best configuration
  Estimate joint probability distribution of the selected configurations
  using UMDA with Laplace correction
  Put a" to the new population
  Repeat{
    Repeat {
      Generate an individual a by sampling joint probability
      distribution
      Compute makespan p of a
    } until p meets Criteria 2
    put a to the new population set
  }until the new population includes n configurations
  Call LocalSearch() to improve the new population
  Replace the old population with the new one
}
return a"

```

Fig. 13: EDA algorithm to find the minimal cost

The neighborhood structure, assigning sequence, and procedure to determine timetable are the same as those described in the Tabu Search section. The local search module used in the EDA algorithm is similar to the one used in the algorithm. We choose UMDA (Univariate Marginal Distribution Algorithm) (Pelikan et al, 1999ab) because the probability k_{ij} of assigning r_j to s_i does not depend on other RMSs in the candidate set. If the best half of the population includes N configurations, k_{ij} increases with the increasing number of appearing $s_i:r_j$ in the selected individuals:

$$k_{ij} = \frac{\sum_{i=1}^N (s_i : r_j) + 1}{N + t_i}, \quad (6)$$

where t_i is the size of the RMS candidate set of sub-job s_i .

The sampling process to create new generation is based on probability k_{ij} . With each sub-job s_i , the RMS r_j is selected following the roulette wheel mechanism as described in Section 3.6. RMS having higher probability has more chance to be selected. Thus, the EDA algorithm tends to choose the RMS that usually appears in high quality configurations.

3.8 Evaluation of Metaheuristic

We have done some experiments to evaluate the performance of standard metaheuristics when applying to the problem of mapping grid-based workflow within an SLA context. Details about the methods and results of the experiment are presented in Section 5. Through the experimental results, we have observed the following:

1. When the size of the problem is relatively small, all metaheuristics would find nearly equal-quality results. We deduce that the found results converge to a value that is very close to the optimal solution.
2. When the size of the problem is big, metaheuristics using local searches such as GLS, ILS, and EDA would find better results than those other metaheuristics which do not use local search.
3. When the size of the problem is big, the runtime of better performance metaheuristics such as GLS, ILS, and EDA is usually exponential. This is a serious drawback when applying the technique to the real system.
4. Except in Tabu Search, most standard metaheuristics use a random mechanism to rapidly explore many different areas in the search space. This mechanism may be able to find good results, but it is very difficult to control the searched area and search process. In contrast, standard Tabu Search searches very carefully the area around the initial configuration and moves only to nearby search areas.

Based on these observations, we conclude that applying directly standard metaheuristics to our problem has many disadvantages, especially the longer runtime. Therefore, it is very desirable to construct new mapping mechanisms, in particular for the situation of heavy communication workflows.

4. Proposed Algorithm for Heavy Communication Workflows

We propose an algorithm called H-Map to map heavy communication workflows onto the grid RMSs (H – stands for heavy). The goal of the H-Map algorithm is to find a solution which satisfies Criterion 2 and costs as little as possible. The overall H-Map algorithm is presented in Figure 14.

First, a set of initial configurations C_o is created. The configurations in C_o should be distributed widely over the search space and must satisfy Criterion 2. If $|C_o|=0$, we can deduce that there is little resource free on the grid and therefore a w-Tabu algorithm (Quan, 2006) is invoked. If w-Tabu cannot find a feasible solution, the algorithm stops. If $|C_o| \neq 0$, the set will gradually be refined to have better quality solutions. The refining

process stops when the solutions in the set cannot be improved. Finally, we have the set C^* . The best solution in C^* is the output as the result of the algorithm. The following sections will describe in detail each procedure in the algorithm depicted in Figure 14.

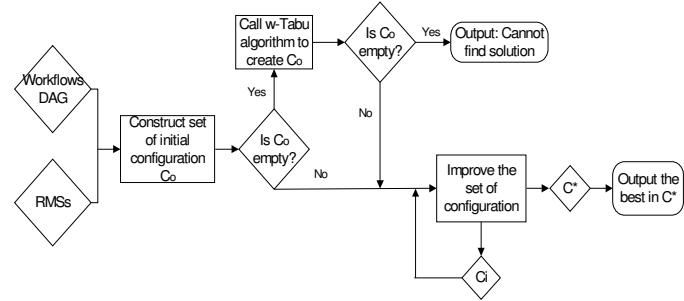


Fig. 14: H-Map algorithm overview

4.1 Constructing the Set of Initial Configurations

The purpose of this algorithm is creating a set of initial configurations which are distributed widely over the search space.

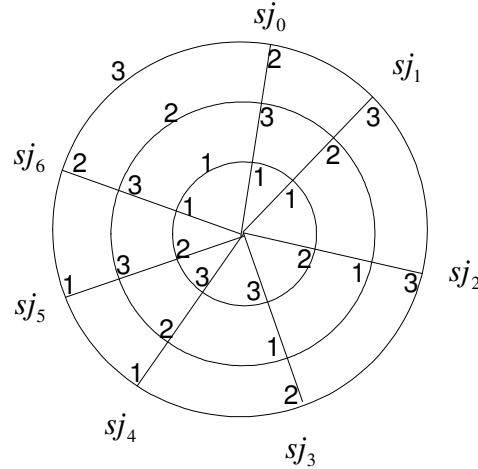


Fig. 15: The configuration space according to cost distribution

Step 0: With each sub-job s_i , we sort the RMSs in the candidate set K_i according to the cost of running s_i . The cost is computed according to Formula 3. The configuration space of the sample is presented in Figure 15 and Table 3. In Figure 15, the RMSs lying along the axis of each sub-job have their costs increasing in the direction from inside to outside. The line connecting each point in every sub-job axis forms a configuration. Figure 15 presents three configurations with increasing indices from inside to outside. It also presents the cost distribution of the configuration space according to Formula 3. The configurations in the outer layers

have greater costs than those in the inner layers. The cost of the configuration lying between two layers is greater than the cost of the inner layer and smaller than that of the outer layer.

Step 1: We pick the first configuration as the first layer in the configuration space. The determined configuration can be presented as a vector. The index of the vector represents the sub-job and the value of the element represents the RMS. The first configuration in our example is presented in Figure 16. Although the first configuration has minimal cost according to Formula 3, we cannot be sure that this is the optimal solution. The real cost of a configuration must consider the neglected cost of data transmission when two sequential sub-jobs are in the same RMS.

Table 3: RMSs candidate for each sub-job in cost order

Sj_ID	RMS	RMS	RMS
sj0	R1	R3	R2
sj1	R1	R2	R3
sj2	R2	R1	R3
sj3	R3	R1	R2
sj4	R3	R2	R1
sj5	R2	R3	R1
sj6	R1	R3	R2

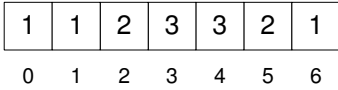


Fig. 16: The first selection configuration of the sample

Step 2: We construct the other configurations using a process similar to the one described in Figure 17. The second solution is the second layer of the configuration space. Then we create a solution having its cost located between layers 1 and 2 by combining the first and second configurations. To do this, we take the first p elements from the first vector configuration and then the next p elements from the second one and repeat the process until we have n elements to form the third one. Thus, we get $(n/2)$ elements from the first vector configuration and $(n/2)$ other elements from the second one. This combination ensures that the target configuration having great difference in cost according to Formula (5) compares favorable to the source configurations. The process continues until reaching the final layer. Thus, we have totally $2*(m-1)$ configurations. With this method in mind, we can ensure that the set of initial configurations is distributed over the search space according to various cost criteria.

Step 3: We check Criterion 2 of all $2*m-1$ configurations. To verify Criterion 2, we have to determine the timetable for all sub-jobs of the workflow. The procedure to determine the timetable of the workflow is similar to the one described in Figure 8. If some of

them do not satisfy requirements specified by Criterion 2, we construct more until we reach enough $2*m-1$ configurations. To do the construction, we change the value of p in the range from 1 to $(n/2)$ in step 2 to create a new configuration.

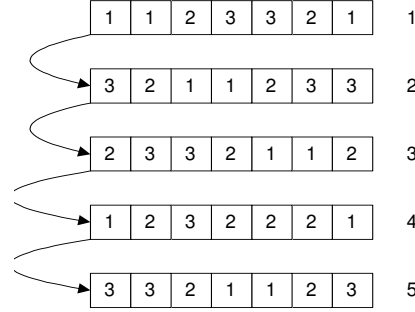


Fig. 17: Procedure to create the set of initial configurations

After this phase we have obtained the set C_0 that includes a maximum of $(2m-1)$ valid configurations.

4.2 Improving Solution Quality Algorithm

To improve the quality of the solutions, we use the neighborhood structure as described in Section 3.2. Let A be the set of neighborhoods of a configuration. The procedure to find the highest quality solution includes the following steps:

Step 1: For all $a \in A$, calculate $cost(a)$ and $timetable(a)$. Pick a^* with the smallest $cost(a^*)$ and satisfying Criterion 2. Add a^* to the set C_1 . Detailed techniques for this step are described in Figure 18.

```

For each subjob in the workflow {
  For each RMS in the candidate list {
    if cheaper then put (sjid, RMS id, improve_value)
    to a list }}
Sort the list according to improve_value
from the begin of the list{
  Compute time table to get the finished time
  If finished time < limit
    break
}
Store the result

```

Fig. 18: Algorithm to improve the solution quality

We consider only configurations having smaller costs than the present configuration. Therefore, instead of computing the cost and the timetable of all configurations in the neighborhood set at the same time, we only compute the cost of each configuration set one at a time. All the configurations are stored in a sorted list. We then compute the timetable of cheaper configurations along the list to

find the first feasible configuration. This technique helps reduce the algorithm's runtime significantly.

Step 2: Repeat Step 1 with all $a \in C_0$ to form C_1 .

Step 3: Repeat Step 1 to 2 until $C_i = C_{i-1}$.

Step 4: $C_i \equiv C^*$. Pick the best configuration in the set C^* .

5. Performance Evaluation

Performance experiment is done with simulation to check for the quality of the mapping algorithms. The hardware and software used in the experiments is rather standard and simple (Pentium 4 2.8Ghz, 2GB RAM, Linux Redhat 9.0, MySQL). The whole simulation program is implemented using programming languages C/C++. With each individual algorithm, we generate several scenarios with different workflow and RMS configurations to fit into the ability of the comparing algorithms. The goals of the experiment are to measure the feasibility, quality of the solution, and time needed for the computation. To do the experiment, 18 workflows with different topologies, number of sub-jobs, sub-job specifications, and amount of data transferred are generated.

These workflows are then mapped to 20 RMSs with different resource configurations and resource reservation contexts by 6 algorithms: H-Map, Simulated Annealing (SA), Guided Local Search (GLS), Iterated Local Search (ILS), Genetic Algorithm (GA) and Estimation of Distribution Algorithm (EDA). The description about resource configurations and workload configurations can be seen at the address: http://it.i.u.de/schools/altmann/DangMinh/desc_expe.txt. The final result of the experiment is presented in Table 4 and in Figures 19 and 20. Column Sjs (Sub-jobs) presents the number of sub-jobs of the workflows used in the experiment. The cost and runtime of solutions generated by each algorithm correlative with each workflow are recorded in column Rt (Runtime) and Cost respectively.

Our experiments are divided into 3 levels of complexity. At the simple level, 7 workflows having number of sub-jobs in the range 7 – 13 are mapped to 20 RMSs. The result shows that the solutions created by different methods converge to one value with each workflow.

At the intermediate level experiment, we use 6 workflows having the number of sub-jobs ranging from 14 to 19. Through this experiment, we observe a difference in the quality of solutions found by different methods. Methods such as GA and SA, which do not use local searched and need relatively less runtime, obtain lower quality solutions than other methods do. Methods such as ILS, GLS, and EDA obtain high quality solutions but need more time than H-Map.

The advanced level experiment uses 5 workflows having number of sub-jobs ranging from 20 to 32. The result of this

experiment shows that H-Map algorithm is able to obtain higher quality solutions with much shorter runtimes than any other algorithms in most cases.

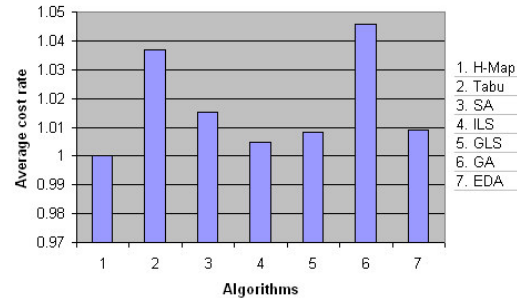


Fig. 19: Cost in relative average value comparison

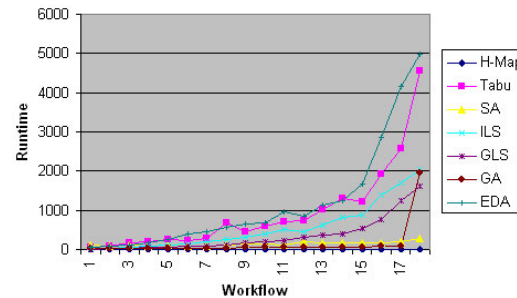


Fig. 20: Runtime comparison

6. Conclusions

In this paper we presented the application of metaheuristics and suggested a proposed algorithm to solve the problem of mapping heavy communication workflows onto grid resources with respect to SLA-defined deadlines and cost optimization. In our work, the distinguished characteristic is that each sub-job of a workflow can be either a sequential or parallel program. In addition, each grid service can handle many sub-jobs at a time. Our performance evaluation showed that the proposed algorithm created solution of equal or better quality than most other metaheuristics and requires significantly less computation time. The latter is a decisive factor for the applicability of any proposed method in real environments, because large-scale workflows can be planned and assigned efficiently with such method.

Table 4: Performance evaluation result

H-Map		Tabu		SA		ILS		GLS		GA		EDA		
Sjs	Rt	Cost	Rt	Cost	Rt	Cost	Rt	Cost	Rt	Cost	Rt	Cost	Rt	Cost
Simple level experiment														
7	1	632.58	58	632.98	105	632.58	21	632.58	14	632.58	25	632.58	62	632.58
8	0.5	756.90	83	757.42	84	756.90	37	756.96	19	756.90	22	756.96	88	756.90
9	1	780.34	157	780.60	114	780.42	54	780.42	29	780.34	27	780.34	126	780.34
10	1	836.18	212	836.63	78	836.18	81	836.18	46	836.18	28	836.18	178	836.18
11	1	892.02	259	926.00	105	892.02	114	892.02	59	892.02	29	892.21	241	892.02
12	2	947.86	235	948.64	90	948.10	147	948.10	80	948.10	36	1005.27	390	947.86
13	1	1003.7	279	1059.77	78	1003.99	201	1003.7	98	1003.99	36	1075.19	462	1003.7
Intermediate level experiment														
14	2	1059.54	669	1114.5	121	1059.89	250	1059.89	127	1059.89	32	1059.89	558	1059.89
15	2	1183.92	453	1247.35	130	1184.21	307	1184.21	167	1184.21	44	1248.86	659	1183.92
16	2	1308.3	607	1352.28	146	1332.53	398	1308.53	187	1308.53	47	1383.53	680	1308.53
17	3	1364.14	696	1433.89	124	1376.42	502	1364.14	222	1377.63	52	1440.81	956	1364.42
18	2	1488.52	744	1575.41	184	1551.74	462	1521.74	302	1501.95	51	1569.39	854	1536.74
19	7	1511.9	1005	1512.04	174	1512.26	620	1512.26	354	1566.09	56	1620.17	1136	1512.26
Advanced level experiment														
20	3	1567.74	1306	1683.23	162	1631.15	815	1567.74	392	1568.15	56	1663.81	1255	1601.15
21	6	1591.12	1225	1713.00	161	1675.67	876	1591.12	524	1621.67	70	1764.18	1663	1621.67
25	5	1785.64	1912	1846.81	180	1871.81	1394	1840.31	763	1843.55	85	1914.91	2845	1830.87
28	7	1888.24	2567	2030.13	197	1960.87	1695	1892.27	1258	1936.83	93	2028.06	4170	1961.30
32	10	2216.22	4545	2350.36	272	2276.33	2046	2283.67	1623	2256.53	1953	2406.97	10976	2276.33

BIOGRAPHIES

Dang Minh Quan received his Ph.D. in Computer Science in 2006 from University of Paderborn, Germany. He received his M.S. and B.E. in Electrical Engineering, in 2003 and 2001, respectively, both from Hanoi University of Technology, Vietnam. Presently he is a Post-doctoral Research Associate at the School of Information Technology, International University in Germany. His research interests include grid computing, parallel/distributed computing, computer network, and network security.

D. Frank Hsu received his B.S. degree from Cheng Kung University (Taiman, Taiwan). M.S. degrees from UTEP and RPI, and Ph.D. degree from the University of Michigan. Dr. Hsu is the Clavius Distinguished Professor of Science and Professor of computer & information sciences at Fordham University, New York, NY. He is also the editor-in-chief of the Journal of Interconnection Networks. His research interests include combinatorics, algorithms, and optimization; network interconnections and communication; informatics and intelligent systems; and information technology and telecommunication infrastructure and implementation strategy.

REFERENCES

- Adam, T. L., Chandy, K. M. and Dickson, J. R. (1974), A comparison of list scheduling for parallel processing systems. *Communication of the ACM*, v. 17 No. 12, pp. 685--690.
- Berriman, G. B., Good, J. C., and Laity, A. C. (2003), Montage: a grid enabled image mosaic service for the National Virtual Observatory. *ADASS*, v. 13, pp. 134--142.
- Brandic I., Benkner, S., Engelbrecht, G. and Schmidt, R. (2005), QoS support for time-critical grid workflow applications. In *Proceedings of e-Science 2005*.
- Brucker, P. (2004), *Scheduling Algorithm*, Third edition, Springer Verlag.
- Burchard, L., Hovestadt, M., Kao, O., Keller, A. and Linnert, B. (2004), The virtual resource manager: an architecture for SLA-aware resource management. In *Proceedings of the IEEE CCGrid 2004*, IEEE Press, pp. 126--133.
- Cao, J., Spooner, D. P., Jarvis, S. A. and Nudd, G. R. (2005), Grid load balancing using intelligent agents. *Future Generation Computer Systems Special Issue on Intelligent Grid Environments: Principles and Applications*, V. 21 No. 1, pp. 135--149.
- Chen, J. and Yang, Y. (2006), Multiple states based temporal consistency for dynamic verification of fixed-time constraints in grid workflow systems. *Concurrency and Computation: Practice and Experience*, April 2006.

- Chen, J. and Yang, Y. (2006), Selecting necessary and sufficient checkpoints for dynamic verification of fixed-time constraints in grid workflow systems. In *Proceedings of the 4th International Conference on Business Process Management(BPM2006)*, Vienna, Austria, LNCS 4102, pp. 445-450.
- Coffman, E. G. (1976), *Computer and Job-Shop Scheduling Theory*. John Wiley and Sons, Inc., New York, NY, 1976.
- Colin, J. Y. and Chretienne, P. (1991), Scheduling with small computation delays and task duplication. *Operation Research*, V. 39 No. 4, pp. 680-684.
- Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Patil, S., Su, M., Vahi, K. and Livny M. (2004), Pegasus : mapping scientific workflows onto the grid. In *Proceedings of the 2nd European Across Grids Conference*, Nicosia, Cyprus, (2004), pp. 28-30.
- Dorigo, M. and Di Caro, G. (1999), The ant colony optimization meta-heuristic. *New Ideas in Optimization*, McGraw-Hill, pp.11-32.
- Dowland, K.A., (1993), Simulated annealing. In C.R. Reeves (ed.) *Modern Heuristic Techniques for Combinatorial Problems*, John Wiley & Sons.
- Gary, M. R. and Johnson, D. S. (1979), *Computers and Intractability: A Guide to the theory of NP-Completeness*. W. H. Freeman and Co.
- Gent, I.P., Macintyre, E., Prosser, P., Smith, B.M., and Walsh, T. (1996), An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In *Proceedings of Principles and Practices of Constraint Programming - CP96*, LNCS 1118, pp. 342--352.
- Gerasoulis, A. and Yang, T. (1992), A comparison of clustering heuristics for scheduling DAG's on multiprocessors. *J. Parallel and Distributed Computing*, v. 16, pp. 276--291.
- Glover, G. (1989), Tabu search: Part I. *ORSA Journal on Computing*, pp. 190--206.
- Glover, F. (1990). Tabu search: Part II. *ORSA Journal on Computing*, v.2, pp. 4--32.
- Hansen, P., and Mladenovic, No. (2001), Variable neighborhood search: principles and applications. *European Journal of Operational Research*, V.130, pp. 97--116.
- Hou, E. S. H., Ansari, N., and Ren, H. (1994), A genetic algorithm for multiprocessor scheduling. In *IEEE Transactions on Parallel and Distributed Systems*, v.5 No.2, pp. 113--120.
- Hovestadt, M. (2003). Scheduling in HPC resource management systems: queuing vs. planning. In *Proceedings of the 9th Workshop on JSSPP at GGF8*, LNCS, pp. 1--20.
- Jansen K., Mastrolilli, M., and Solis-Oba, R. (2000), Approximation algorithms for flexible job shop problems. In *Proceedings of Latin American Theoretical Informatics (LATIN'2000)*, LNCS 1776, pp. 68--77.
- Kacem, I., Hammadi, S., and Borne, P. (2001). Approach by localization and multi-objective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems, Man, and Cybernetics*. Part C, V. 32 No. 1, pp. 1-13.
- Kacem, I., Hammadi, S., and Borne, P. (2002), Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. *Journal of Mathematics and Computers in Simulation*, Elsevier.
- Kirkpatrick, S., Gelatt, C. and Vecchi, M. (1983). Optimization by simulated annealing. *Science*, V. 220, pp. 671--680.
- Kohler, W. H. and Steiglitz, K. (1974), Characterization and theoretical comparison of branch-and-bound algorithms for permutation problems. *Journal of ACM*, V. 21 No. 1, pp. 140-156.
- Kruatrachue, B. and Lewis, T. G. (1987), *Duplication Scheduling Heuristics (DSH): A New Precedence Task Scheduler for Parallel Processor Systems*. Oregon State University, Corvallis, OR.
- Kruatrachue , B., and Lewis, T. (1988), Grain size determination for parallel processing. *IEEE Software*, V.5 NO.1, pp. 23--32.
- Kumar, V. (1992). Algorithms for constraint-satisfaction problems: a survey. *AI Magazine*, V. 13 NO. 1, pp. 32--44.
- Kwok Y. K. and Ahmad, I. (1999), Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)*, V. 31 No. 4, pp. 406--471.
- Lovas, R., Dózsa, G., Kacsuk, P., Podhorszki, N., and Drótos, D. (2004), workflow support for complex grid applications: integrated and portal solutions. In *Proceedings of 2nd European Across Grids Conference*, Nicosia, Cyprus, pp. 136-144.
- Ludtke, S., Baldwin, P. and Chiu, W. (1999), EMAN: semiautomated software for high-resolution single-particle reconstruction. *Journal of Structure Biology*, V. 128, pp. 146-157.
- Mastrolilli, M., and Gambardella, L.M. (2000), Effective neighborhood functions for the flexible job shop problem. *Journal of Scheduling*, V. 3 No. 1, pp. 3--20.
- McGough, S., Afzal, A., Darlington, J., Furmento, N., Mayer, A. and Young, L. (2005), Making the grid predictable through reservations and performance modelling. *The Computer Journal*, V.48 No.3, pp. 358--368.
- Nowicki, E. and Smutnicki, C. (1996), A fast taboo search algorithm for the job shop problem. *Management Science*, V. 42, pp. 797--813.
- Nowicki, E., and Smutnicki, C. (2005) An advanced taboo search algorithm for the job shop problem. *Journal of Scheduling*, pp. 145--159.
- Pelikan, M., Goldberg, D. and Cantú-Paz, E. (1999a), BOA: the bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99* (Orlando, Fla.), pp. 525-532.
- Pelikan, M., Goldberg, D. and Lobo, F. (1999b), A survey of optimization by building and using probabilistic models. *Tech. Rep. No. 99018, IlliGAL*, University of Illinois.
- Quan, D.M., and Kao, O. (2005a), On architecture for an SLA-aware job flows in grid environments. *Journal of*

- Interconnection Networks*, World Scientific Publishing Co., pp. 245--264.
- Quan, D.M. and Kao, O. (2005b). SLA negotiation protocol for grid-based workflows. In *Proceedings of the International Conference on High Performance Computing and Communications (HPPC-05)*, LNCS 3726, pp. 505--510.
- Quan, D.M., and Kao, O. (2005c), On architecture for an SLA-aware job flows in grid environments. In *Proceedings of the 19th IEEE International Conference on Advanced Information Networking and Applications (AINA 2005)* , IEEE Press , pp. 287--292.
- Quan, D.M., and Kao, O. (2005d), Mapping grid job flows to grid resources within SLA context. In *Proceedings of the European Grid Conference, (EGC 2005)*, LNCS 3470, pp. 1107--1116.
- Quan, D.M. (2006), Error recovery mechanism for grid-based workflow within SLA context, *To appear in International Journal of High Performance Computing and Networking (IJHPCN)*.
- Quan, D.M., and Hsu, D.F. (2006), Network-based resource allocation for Grid Computing within an SLA context. In *Proceedings of the 5th International Conference on Grid and Cooperative Computing, (GCC 2006)*, IEEE press, pp.
- Rewini , H. E. and Lewis, T. G. (1990), Scheduling parallel program tasks onto arbitrary target machines. *Journal of Parallel and Distributed Computing*, V.9 No.2, pp. 138--153.
- Richter, L. (2004), Workflow support for complex grid applications: integrated and portal solutions. In *Proceedings of the 2nd European Across Grids Conference*, pp. 247-256.
- Sadeh, N. and Fox, M.S. (1996), Variable and value ordering heuristics for the job shop constraint satisfaction problem. *Artificial Intelligence*, V. 86, pp. 1--41.
- Sahai, A. (2002), Automated SLA monitoring for web services, *DSOM 2002*, LNCS, Springer Verlag, 2002 pp. 28--41.
- Sarkar, V. (1989), *Partitioning and Scheduling Parallel Programs for Multiprocessors*. MIT Press, Cambridge, MA.
- Shahid, A., Muhammed, S. T. and Sadiq, M. (1994), GSA: scheduling and allocation using genetic algorithm. In *Proceedings of the Conference on European Design Automation*, September , 1994, pp. 84-89.
- Sih, G. C., and Lee, E. A. (1993), Declustering: a new multiprocessor scheduling technique. In *IEEE Transactions on Parallel and Distributed Systems*, V.4, No.6, pp. 625--637.
- Spooner, D. P., Jarvis, S. A., Cao, J., Saini, S. and Nudd G. R. (2003), Local grid scheduling techniques using performance prediction. In *IEEE Proceedings - Computers and Digital Techniques*, 150(2), pp. 87--96.
- Stützle, T. (1999a), Iterated local search for the quadratic assignment problem. *Tech. Rep. Aida-99-03*, FG Intellektik, TU Darmstadt.
- Stützle, T. (1999b), *Local Search Algorithms for Combinatorial Problems---Analysis, Algorithms and New Applications*. DISKI---Dissertationen zur Künstliken Intelligenz. infix, Sankt Augustin, Germany.
- Voudouris, C. (1997), *Guided Local Search for Combinatorial Optimization Problems*. PhD dissertation, Department of Computer Science, University of Essex.
- Voudouris, C. and Tsang, E. (1999), Guided local search, *European Journal of Operation Research*, V. 113 No.2, pp. 469--499.
- Wolski, R. (2003), Experiences with predicting resource performance on-line in computational grid settings. *ACM SIGMETRICS Performance Evaluation Review*, V. 30 NO. 4, pp. 41--49.
- Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., and Chang, H. (2004), QoS-aware middleware for web services composition. In *IEEE Transactions on Software Engineering*, V.30 No.5, pp. 311--327.