

## Resource allocation algorithm for light communication Grid-based workflows within an SLA context

Dang Minh Quan †\* & Jörn Altmann ‡

†‡ School of Information Technology, International University in Germany, Germany

Email: †quandm@upb.de, ‡jorn.altmann@acm.org

*(v3.0 released November 2004)*

Service Level Agreements (SLAs) are currently one of the major research topics in Grid Computing. Among many system components for supporting SLA-aware Grid-based workflow, the SLA mapping mechanism receives a prominent position. It is responsible for assigning sub-jobs of the workflow to Grid resources in a way that meets the user's deadline and minimizes costs. Assuming many different kinds of sub-jobs and resources, the process of mapping a light communication workflow within an SLA context defines an unfamiliar and difficult problem. This paper presents a mapping algorithm, which can cope with this problem. The quality and efficiency of the algorithm is demonstrated through performance measurements.

*Keywords:* Grid computing, light communication workflow, mapping, Service Level Agreement

### 1 Introduction

Service Level Agreements (SLAs) are currently one of the major research topics in Grid Computing. We have developed the system for supporting SLA for the Grid-based workflow [1–6] in which our mapping problems are distinguished from other works [7, 8] in many aspects. In our system, sub-jobs of the workflow can be either sequential or parallel programs and each Grid site can handle many sub-jobs simultaneously. Our ideas about Grid-based workflow mapping within the SLA context have 3 main scenarios.

- Mapping Grid-based workflow within SLA context with execution time optimization. This case applies mainly for the error recovery phase [3].
- Mapping heavy communication Grid-based workflow within the SLA context, satisfying the deadline and optimizing the cost [4].
- Mapping light communication Grid-based workflow within the SLA context, satisfying the deadline and optimizing the cost. The key difference in this case compared to the above case is that we can assume the data transfer takes place in only one time slot. We can also omit the data transfer cost. Thus, we can apply several dedicated techniques for solving the problem. An initial work for the case of workflows with light communication is presented in [1], but without considering the variety of sub-job's execution time with different resource configurations. This paper, which belongs to a series of efforts to develop a framework supporting SLAs for Grid-based workflows [1–6], will present an approach that fills this gap.

---

\*Corresponding author

Table 1. Resource requirements for sub-jobs

Sj_ID	CPU	Storage	Exp	Runtime
0	64	59	1	18
1	48	130	2	16
2	64	142	1	17
3	64	113	2	20
4	48	174	1	18
5	48	97	1	19
6	64	118	1	18

Like many popular system handling Grid-based workflow [9–11], our proposed workflow system is of the Directed Acyclic Graph (DAG) form. In the case of light communication workflows, the data to be transferred among sub-jobs is very small, usually less than 10MB. The user is also required to specify the estimated execution time of each sub-job together with the specific resource requirements. The time is split into slots with each slot representing a constant period of real time, usually from 3 to 5 minutes. Figure 1 illustrates an example of a Grid workflow with resource requirements as is specified in Table 1. The number above each link of the workflow represents the amount of data to be transferred among sub-jobs.

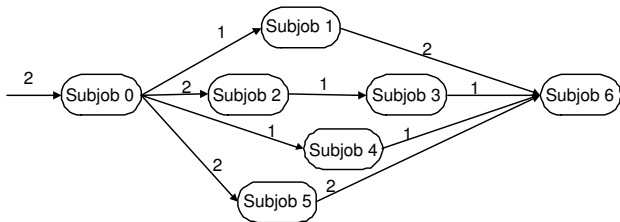


Figure 1. A sample workflow

In each Grid site, which is High Performance Computing Center (HPCC), the resources are managed by the software called local Resource Management System (RMS). Each RMS has its own unique resource configuration. To ensure that the sub-job can be executed within a dedicated time period, the RMS must support advance resource reservation such as CCS [12]. In our system, RMS has the capability of reserving three main types of resources: CPUs, storages and experts. An extension to other devices and other related resources is straightforward. For our example, we assume to have three RMSs with the same number of CPU =96, number of storage =3TB, number of expert =10. All reservation profiles are empty.

HPCCs usually connect with the network through a broadband link which is greater than 100Mbps. The length of one time slot in our system is between 2 and 5 minutes. Thus, the amount of data to be transferred through a link in one time slot can be from 1.2GB to 3GB. With the small amount of data to be transferred among sub-jobs of light communication workflow, the data transfer can happen easily in one time slot (right after the sub-job finished its calculation) without affecting any other communication between two RMSs.

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

- Let  $R$  be the set of all grid RMSs. This includes a finite number of RMSs, which provide static information about controlled resources and the current reservations/assignments.
- Let  $S$  be the set of all sub-jobs in a given workflow. This includes all sub-jobs with the current resource and deadline requirements.
- Let  $E$  be the set of all data transfers in a workflow. This indicates the dependency between the sub-jobs and the necessity for data transfers between the sub-jobs.
- Let  $K_i$  be the set of resource candidates for sub-job  $s_i$ . This 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. The required solution is a set defined in Formula 1.

$$M = \{(s_i, r_j, start\_slot) | s_i \in S, r_j \in K_i\} \quad (1)$$

If the solution does not have *start\_slot* for each  $s_i$ , it becomes a configuration as defined in Formula 2.

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

A feasible solution must satisfy following conditions:

- **Criteria1:** All  $K_i \neq \emptyset$ . There is at least one RMS in the candidate set of each sub-job.
- **Criteria2:** The total execution time period of the workflow must be within the expected period given by the user.
- **Criteria3:** The dependency among the sub-jobs is resolved and the execution order remains unchanged.
- **Criteria4:** Each RMS provides a profile of currently available resources and can run many sub-jobs of a single workflow both sequentially and 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, with each time slot in the profile of available resources and profile of resource requirements, the number of available resources must be larger than the resource requirement.
- **Criteria5:** The data transmission task  $e_{k_i}$  from sub-job  $s_k$  to sub-job  $s_i$  takes one timeslot right after the finished time of sub-job  $s_k$ ,  $e_{k_i} \in E$ . If sub-job  $s_k$  and sub-job  $s_i$  are executed in the same RMS the data transfer task is neglected.

In the next phase, feasible solutions with the lowest cost are sought. As the number of data to be transferred among sub-jobs in the workflow is very small, we can omit the cost of data transfer. Thus, the cost  $C$  of a grid workflow is defined in Formula 3. It is a sum of the cost of using: (1) the CPU, (2) storage and (3) expert knowledge.

$$C = \sum_{i=1}^n s_i.r_t * (s_i.n_c * r_j.p_c + s_i.n_s * r_j.p_s + s_i.n_e * r_j.p_e) \quad (3)$$

with  $s_i.n_c, s_i.n_s, s_i.n_e$  are number CPU, number storage, number expert of sub-job  $s_i$  respectively.  $s_i.r_{t_j}$  is the execution time of sub-job  $s_i$  in RMS  $r_j$ . The value of  $s_i.r_{t_j}$  can be determined with the mechanism described in [13].  $r_j.p_c, r_j.p_s, r_j.p_e$  is the price of using CPU, storage, expert of RMS  $r_j$  respectively.

It can be easily shown that the optimal mapping of a workflow onto grid-based RMSs with optimized cost is an NP-hard problem.

## 2 Related works

The mapping of jobs to suitable resources is one of the core tasks in Grid Computing. However, the majority of the research has been to the mapping of singular jobs, which do not exhibit dependencies to other jobs regarding input/output data. The mapping of workflows, where a single job is divided into several sub-jobs, is the next research step. In the literature, there are many attempts at this issue such as [9–11, 13]. However, all those mechanisms work to map a workflow to the Grid resources in the best effort manner.

The mapping of Grid workflows onto Grid resources based on existing planning technology is presented in [9]. This work focuses on coding the problem to be compatible with the input format of specific planning systems and thus transfers the mapping problem to a planning problem. Although this is a flexible way to gain different goals, significant disadvantages regarding the huge resource usage and long response times. For example, with a workflow including 20 sub-jobs and

the Grid including 10 RMSs, a planning system cannot solve the problem on a desktop computer because of not enough memory [1]. If the size of the problem is slightly smaller, the planning system needs several hours to find the high quality solution. This is the main cause that a planning system should not be used for a business broker.

In two separated works [7, 8], Zeng et al and Iwona et al built systems to support QoS features for Grid-based workflow. In their work, a workflow includes many sub-jobs, which are sequential programs, and a Grid service has the ability to handle one sub-job at a time. To map the workflow on to the Grid services, they used the Integer Programming method. Applying Integer Programming to our problem is impossible. The reason is that an RMS can handle many parallel programs at a time. Thus, we do not know how many, which, and when sub-jobs will be run in an RMS and we cannot present the constraint of available capacity as described in Criteria 4.

Our problem has a close relation to the classic job shop scheduling problem (JSSP) [14]. 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. As the job shop scheduling problem is an NP hard problem, two main methods to solve this problem – the complete and the incomplete method – exist. A complete method explores systematically, though very often implicitly, the whole search space. To do this, most complete methods construct in a "step by step" way a solution and then backtrack in case of failure [15]. These methods usually use various heuristics [15–17] to guide the choice of the next variable to be instantiated and its value, and 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 [15] and try to eliminate heuristically as many as possible solutions not leading to an optimum one.

Complete and exact methods have in general exponential time complexity. Therefore, the solving time required by such a method may consequently become prohibitive for large-sized problems.

An incomplete (non-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. In fact, they constitute a very interesting alternative for the practical solving of many hard and large-sized problems. The well-known methods include Tabu Search [18, 19], Simulated Annealing [20], GA [21] and so on.

In the literature, when applying local search for 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 N1, N2, N3, N4 [14] and N5 [22] have been proposed. The primary activity of these neighborhoods is changing the process sequence of two operations in the same machine. However, while 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 simultaneously. Thus, in our solution such process sequence changing does not exist.

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 it. The FJSSP is far more complicated than the classical JSSP with two main problems, that of assigning each operation to an appropriate machine, and that of sequencing the operation in each machine. To solve the FJSSP, the research community applied several local search methods and proposed several techniques [23, 24] to decrease the huge search space. Those techniques, based strictly on the character of each machine, can only do one operation at a time. Our problem differs from the FJSSP 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, we cannot apply the proposed techniques to our problem because of the characteristic differences.

Related to mapping task graph to resources, there is also the multiprocessor scheduling precedence-constrained task graph problem [25, 26]. As this is a well-known problem, the literature has recorded a lot of methods for this issue, which can be classified into several groups [27]. The classic approach is based on the so-called list scheduling technique [28, 29]. More recent approaches are UNC Scheduling [30, 31], BNP Scheduling [28, 33, 35], TDB Scheduling [34, 36], APN Scheduling [37], and genetic [38, 39]. 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 this problem is a strictly sequential program. Each node in this problem can process one task at a time while each RMS in our problem can process several sub-jobs at a time. For those reasons, applying a local search approach for this

problem is very rare. In our solution, a specific local search algorithm is used and has proven to be very efficient.

In the SLA context, the problem of mapping a workflow to Grid resources is greatly different from the formal one, both in the character of the sub-job and character of the Grid resource in RMSs as presented in Section 1. An initial mechanism based on Tabu search to map a light communication workflow onto Grid resources is described in [1]. In order to shorten the computation time caused by the high number of resource profiles to be analyzed and by the flexibility of the sub-jobs' execution time, several techniques for reducing the search space are introduced. However, in this paper these techniques cannot be applied to solve the problem because the variation of sub-job's execution time in different RMSs is not considered. This key character leads to the following differences between the L-Map algorithm and the L-Tabu algorithm.

- If the CPU speed is higher, the price is higher. As the execution time is constant, L-Tabu algorithm tends to select the CPU with equal CPU speed. It does not pay attention to the case that the higher CPU speed makes the execution time of sub-jobs shorter and thus the total cost is lower even the price is higher. The L-Map algorithm addresses this issue.
- Moreover, when RMSs having CPU speed equal the requirement are busy and we have to consider the RMSs having CPU speed higher than the requirement, it is obvious that the cost of running a sub-job with originally estimated execution time is larger than running with a reduced execution time. The L-Tabu algorithm does not address this case. In addition, the L-Tabu algorithm has less chance to find a suitable period in the resource reservation profile of an RMS than the L-Map algorithm. This is because the required execution time period in the L-Map could be shorter and thus, easier to find a suitable gap in the reservation profile. This factor also helps to reduce the cost of the L-Map algorithm.
- Assuming a constant execution time, L-tabu can build the global relation profile between the workload and the Grid resource. From this profile, it can move the sub-job to the period having many free resources. With time variation, this technique cannot be done because the workload profile varies depends on the resource. Thus, there is no global relation profile in the L-map algorithm.

In [4], we described an algorithm called H-Map to map heavy communication Grid-based workflow within the SLA context. The key difference between a light communication workflow and a heavy communication workflow is the amount of data transfer between sub-jobs. This character influences the mapping algorithm in following aspect. With the heavy communication workflow, the cost of data transfer contributes a major part to the total cost of running the workflow. A solution with expensive RMSs but neglecting the data transfer may be cheaper than a solution with cheap RMSs but without omitting the data transfer. Thus, we cannot eliminate the solution space of the heavy communication workflow. There is no elimination of the solution space in H-Map algorithm. In contrast, we can omit the cost of data transfer of the light communication workflow in the L-map algorithm. This allows us to form a sorted solution space. If a solution is found in the lower part of the solution space, the upper part of it can be moved out. This technique helps to reduce the runtime of the L-Map algorithm.

Recently appearing algorithms such as [40, 41] also map a workflow onto Grid resource with minimum of cost. However, those works assume Grid resource service handling mechanisms, which can only handle one sub-job at a time. These assumptions are not realistic, since many High Performance Computing Centers (HPCC) provide their entire computing services under one single Grid resource service [42]. We adapted all those algorithms to our problem as a mean of performance comparison. A detailed description of those approaches that apply to our problem can be found in the following sections.

## 2.1 *H-Map algorithm*

The work in [4] solved the problem of mapping a heavy communication workflow onto the Grid resources. An algorithm call H-Map is proposed. The main idea of the H-Map algorithm is forming a widely distributed set of initial configurations and doing the local search with each of them to find the best solution. As the problem in [4] is close to the problem in this paper, we can easily adapt the H-Map algorithm to map the light communication workflow onto Grid resources. The framework of the algorithm is retained as described in Figure 2. We change only the computing timetable function and computing cost function to suit the new requirement.

```

Sort the solution space according to the computation cost
Clear the initial set of solutions
While not enough solutions {
  Form new configuration by combining 2 cost levels
  Compute timetable to check the feasible
  If feasible, put to the initial set of solutions
}
For each solution in the set {
  Do local search with the cost function
}
Pick the best solution

```

Figure 2. Framework of the H-Map algorithm

## 2.2 DBC algorithm

The original DBC Grid scheduling algorithm [40], called the cost-time optimization scheduling algorithm, is used to schedule parameter sweep application on global Grids. The algorithm builds on the cost-optimization and time-optimization scheduling algorithms. This is accomplished by applying the time-optimization algorithm to schedule task-farming jobs on distributed resources having the same processing cost. Even though this algorithm only supports sequential task, the idea can be applied to our problem since our workflow can be considered as a parameter sweep application. The application of this strategy to the problem is presented in Figure 3.

```

Analyze the workflow into set of sub-jobs in sequential
layers. Sub-jobs in the same layer do not depend on each
other.
For each sub-job in the set {
  Sort the candidate RMSs according to cost order.
  For each RMS in the sorted candidate list {
    calculate the execution time of that sub-job on the
    RMS. If it meet the deadline then assigned the sub-
    job to the RMS
  }
}

```

Figure 3. The application of DBC algorithm to our problem

## 2.3 Genetic Algorithm (GA)

GA [21] is a part of Evolutionary Computing, which is inspired by Darwin's theory of evolution. GA begins with a set of solutions called population. Solutions from one population are selected according to their fitness and used for forming a new population. This is motivated by a hope that the new population will be better than the old one. This process is repeated to find out the best solution. In [41], Yu et al also use GA algorithm to map a workflow onto the Grid resources. However, the work in [41] considered each service handling only one sub-job at a time. The GA algorithm to find the minimal cost of a light communication workflow within SLA context is presented in Figure 4.

Parents are selected according to the roulette wheel method. The fitness of each configuration =  $1/\text{cost}$ . The cost value is computed as described in Section 1. Firstly, the sum  $L$  of all configuration

```

Repeat{
  Repeat {
    Pick randomly an initial configuration a
    Compute finished_time p of a
  } until p meets Criteria 1
  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 cost
      Crossover the parent with a probability to form new configuration
      Mutate the new configuration with a probability
      Compute finished_time p of the new configuration
    } until p meets Criteria 1
    Put the new configuration to the new population }
  }
return a"

```

Figure 4. GA algorithm to find the minimal cost

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 to the configuration where we are.

The crossover point is chosen randomly. The offspring is formed by copying from two parts of the configurations. The mutation point is chosen randomly. At the mutation point,  $r_j$  of  $s_i$  is replaced by another RMS in the candidate RMS set. It is noted that the probability to have mutation with an offspring is low, ranging approximately from 0.5% to 1%. After having the new configuration  $a'$  by choosing random configuration, doing crossover or doing mutation, the module computing timetable is called on to check if the finished time of  $a'$  is within the user's limitation. If the finished time of  $a'$  exceeds the user's limitation, a new configuration will be selected.

### 3 L-Map algorithm

In this paper, we propose an algorithm called L-Map to map a light communication workflows onto the grid RMSs (L - stands for light). The goal of the L-Map algorithm is to find a solution which satisfies Criterion 1-5 and is as inexpensive as possible.

Each sub-job has different resource requirements about the type of RMS, the type of CPU and so on. There are a lot of RMSs with different resource configurations. The initial action is finding among those heterogeneous RMSs the suitable RMSs, which can meet the requirement of the sub-job. The matching between the sub-job's resource requirement and the RMS's resource configuration is done by several logic checking conditions in the WHERE clause of the SQL SELECT command. This work will satisfy Criteria 1. If each sub-job has  $m$  RMSs in the candidate list, we could have  $m^n$  configurations.

The overall L-Map algorithm to map DAG to resources is presented in Figure 5. The first phase of the algorithm is trying to find out a high quality and feasible solution. We start by assigning each sub-job to the cheapest RMS. If there is a conflict in an RMS, the conflict period is resolved by adjusting the earliest start time or the latest stop time of the sub-job. After resolving this, if the conflict period still exists, sub-jobs joining the period are moved to other RMSs. If the movement is not successful we can deduce that the Grid has few free resources and the w-Tabu algorithm [3] is invoked. If w-Tabu cannot find an initial feasible solution, the algorithm will stop. We repeat the process of resolving conflict periods until there is no conflict period or the algorithm stops. After phase 1, we have a feasible solution. Starting from this solution, we limit the solution space by

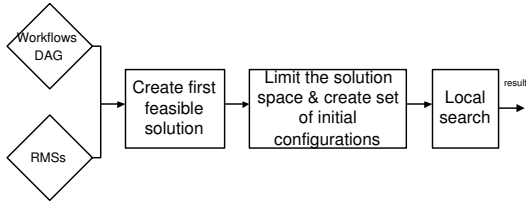


Figure 5. Framework of the L-Map algorithm

removing expensive RMSs. After that, we form a set of initial configurations spreading over the search space. A local search is invoked to improved the quality of the configurations and best solution selected. The following sections will describe in detail each procedure in the algorithm.

### 3.1 Creating the initial feasible solution

The sequence of steps in the procedure of creating the initial feasible solution is presented in Figure 6.

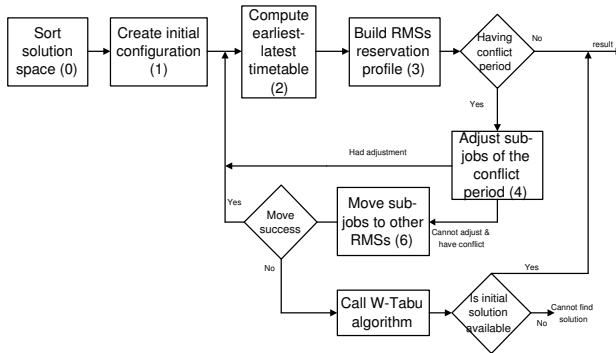


Figure 6. Procedure to create the initial solution

**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 using Formula 3. The RMS having the lower cost to run the sub-job locates at the lower position. The solution space of the sample is presented Figure 7. Each box RMS-Rt presents the RMS and the execution time of the sub-job in this RMS.

R2 - 16	R3 - 14	R3 - 17	R2 - 17	R3 - 15	R2 - 16	R2 - 17
R3 - 17	R2 - 14	R2 - 16	R3 - 15	R2 - 17	R3 - 16	R3 - 16
R1-16	R1 - 14	R1 - 16	R1 - 16	R1 - 15	R1 - 14	R1 - 16
sj0	sj1	sj2	sj3	sj4	sj5	sj6

Figure 7. The solution space in cost order

**Step 1:** We form the first configuration by assigning each sub-job to the RMS having the lowest cost in the candidate list. 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 8.



Figure 8. The first selection configuration of the example

Table 2. The earliest-latest timetable

Sj_ID	Earliest start	Latest stop
0	10	37
1	26	69
2	26	41
3	42	69
4	26	69
5	26	69
6	58	85

**Step 2:** As the execution time of each sub-job in the determined RMS is defined and the time to do data transfer is fixed, we can compute the earliest start time and the latest stop time of each sub-job using the conventional graph algorithm. In our example, assume that the user wants the workflow to be started at time slot 10 and stopped at time slot 85, the earliest-latest timetable is presented in Table 2.

**Step 3:** From the data of the earliest-latest timetable, with each RMS appearing in the configuration and each type of reserved resource in the RMS, we build the resource reservation profile. In this step, the execution time of the sub-job is computed from the earliest start time to the latest stop time. The built profiles could have many conflict periods, in which, the number of required resource is greater than the available resource. If we do not resolve those periods, we will not have a feasible solution. In this algorithm, CPU, storage, and expert are considered in the same way. Each resource has its own reservation profile. The characters of each profile are very similar to each other. To have a feasible solution, we have to resolve the conflict in three profiles. As they are very similar to each other, in this paper, we present only the CPU profile to demonstrate the idea. In our example, only RMS1 appears in the configuration. The CPU reservation profile of the RMS 1 is presented in Figure 9.

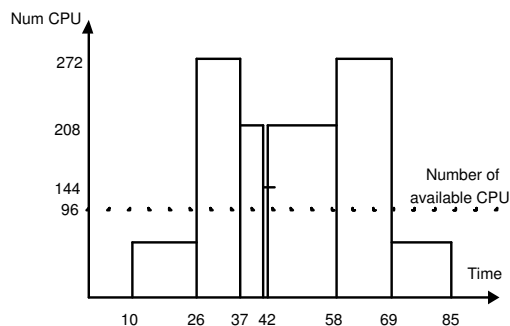


Figure 9. Reservation profile of RMS 1

**Step 4:** There are many sub-jobs joining the conflict period. If we move sub-jobs out of it, the conflict rate will be reduced. The movement is performed by adjusting the earliest start time or the latest stop time of the sub-jobs and thus, the sub-jobs are moved out of the conflict period. One possible solution is shown in Figure 10(a), where either the latest stop time of sub-job1 is set to  $t_1$  or the earliest start time of sub-job2 is set to  $t_2$ . The second way is to adjust two sub-jobs simultaneously as depicted in Figure 10(b). A necessary prerequisite here is that after adjustment, the latest stop time minus the earliest start time of the sub-job is larger than its execution time.

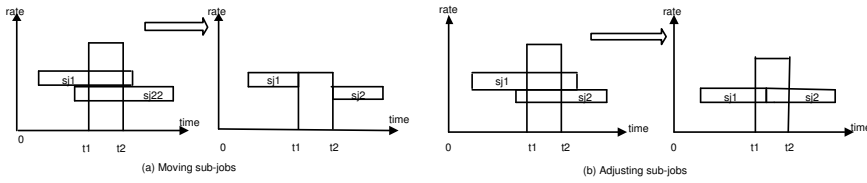


Figure 10. Resolving the conflict period

**Step 5:** We adjust the earliest start time and latest stop time of the sub-jobs relative to the moved sub-jobs to ensure the integrity of the workflow. Then we repeat steps 3 and 4 until we cannot adjust the sub-jobs further. In our example, after this phase, the CPU reservation profile of RMS 1 is as presented in Figure 11.

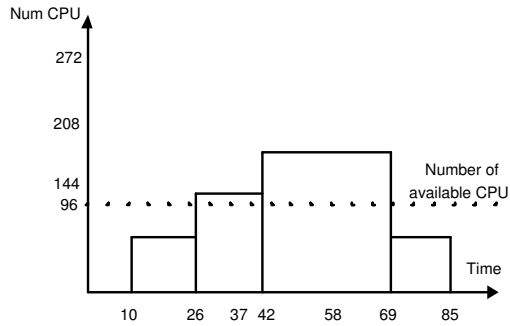


Figure 11. Reservation profile of RMS 1 after adjusting

**Step 6:** If after the adjusting phase, there are still some conflict periods, we have to move some sub-jobs contributing to the conflict to other RMSs. The resource in the RMS having the conflict period should be allocated as much as possible, so that the cost for using resources will be kept at a minimum. This is a knapsack problem, which is known to be NP-hard. Therefore, we use an algorithm as presented in Figure 12. This algorithm ensures that the remaining free resource after filling phase is always smaller than the smallest sub-job. If a sub-job cannot be moved to another RMS, we can deduce that the Grid resource is busy and then the w-Tabu algorithm [3] is invoked. If w-Tabu cannot find an initial feasible solution, the algorithm will stop.

```

Select the most serious conflict period
Determine all sub-jobs contributing to the period
Sort those sub-jobs according to cost in descend order
For each sub-job in the list {
  If the resource free greater than the resource required by
the sub-job
  Let the sub-job stay in the RMS
  Update the number of resource free of the period
Else
  Assign the sub-job to the next RMS in its sorted
candidate list
}

```

Figure 12. Moving sub-jobs algorithm

In our example, the most conflict period is 42-69 with the contribution of sub-job 3, 5 and 4 sorting in descending order according to the cost. We can fill the period with sub-job 3 while sub-job 4 is moved to RMS 2 and sub-job 5 is moved to RMS 3. After this step, we have a new configuration as presented in Figure 13.

1	1	1	1	2	3	1
0	1	2	3	4	5	6

Figure 13. The new configuration of the example

**Step 7:** With this new configuration, the process from step 3 to step 6 is repeated until there is no conflict period. After this phase, we have a feasible candidate solution. The feasible solution of our example is depicted in Figure 14.

1	2	1	1	2	3	1
0	1	2	3	4	5	6

Figure 14. The first feasible solution of the example

### 3.2 Limiting the solution space

Suppose that each sub-job has  $m$  candidate RMSs. Suppose that in the feasible solution, the RMS has the highest ranking at  $k$ . Thus, with each sub-job, we remove all its candidate RMSs having a rank greater than  $k$ . The process which applies to our example is presented in Figure 15. We limit the solution space in this way for two reasons:

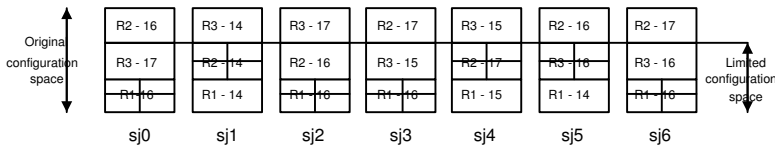


Figure 15. The limited solution space

- The lower area of the solution space contains the more inexpensive solutions for sub-jobs. Therefore, the ability to have high quality solutions in this area is very great and it should be considered intensively. In contrast, the higher area of the solution space contains expensive solutions for sub-jobs; thus, the ability to have high quality solution in this area is very low. For that reason, to save the computation time, we can by pass this area.
- The selected solution space contains at least one feasible solution. Thus, we can be sure that with this new solution space we can always find out an equal or better solution than the one found.

### 3.3 Creating the set of initial configurations

The set of initial configurations should be distributed over the solution space as widely as possible. Therefore, we create the new configuration by shifting onward to the first feasible solution. Assume each sub-job of having  $k$  candidate RMSs: we will shift  $(k - 1)$  times to create  $(k - 1)$  configurations. Thus, there are  $k$  configurations in the initial set including the found feasible

solution. With our example, the procedure is expressed in Figure 16. It is noted that in this step, a configuration can be either feasible or unfeasible.

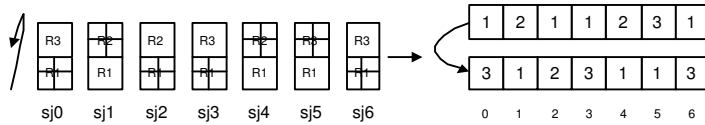


Figure 16. Creating the set of initial configurations

### 3.4 Local search

A local search procedure is used to find a feasible solution and to improve the quality of the solution as far as possible starting from a configuration in the initial set. The overall local search procedure is presented in Figure 17.

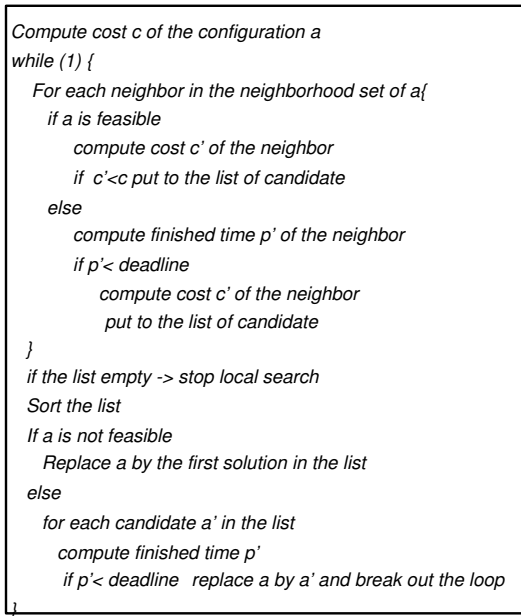


Figure 17. The local search procedure

If the initial configuration is not feasible, we search in the neighborhood of the candidate configuration for the feasible solutions satisfying Criterion 2. Then we replace the initial one by the highest quality solution found. In this case we have to compute the timetable and check the deadline of all configurations in the neighborhood.

If the initial configuration is feasible, we consider only configurations having lower costs than the present solution. 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 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 as compute timetable procedure need a lot of time to complete. The compute timetable procedure and compute cost procedure is similar to the one in [4] with only small changes to suit the light communication scenario. The time to perform the local search procedure varies depending on the number of invoking module computing timetable. If this number is small the computation time will be short and vice versa.

## 4 Performance experiment

The goal of the experiment is to measure the feasibility of the solution, its cost and the time needed for the computation. The environment used for the experiments is rather standard and simple (Intel Duo 2,8Ghz, 1GB RAM, Linux FC5).

To do the experiment, we generated 20 different workflows which:

- Have different topologies.
- Have a different number of sub-jobs from 21 to 32.
- Have different sub-job specifications. Without loss of generality, we assume that each sub-job has the same CPU performance requirement.
- Have different amounts of data transfer.

The expected deadline of each workflow is equal to the earliest finished time of the workflow, which can be calculated easily by using a conventional graph algorithm. The execution time of each sub-job in each type of RMS is assigned by using formula 4.

$$rt_j = \frac{rt_i}{\frac{pk_i + (pk_j - pk_i) * k}{pk_i}} \quad (4)$$

With  $pk_i$ ,  $pk_j$  is the performance of a CPU in RMS  $r_i$ ,  $r_j$  respectively and  $rt_i$  is the estimated execution time of the sub-job with the resource configuration of RMS  $r_i$ .  $k$  is the speed up control factor. Within the performance experiment, in each workflow, 60% of the number of sub-jobs have  $k = 0.5$ , 30% of the number of sub-jobs have  $k = 0.25$ , 10% of the number of sub-jobs have  $k = 0$ . As the difference in the static factors of an RMS such as OS, CPU speed and so on can be easily filtered by SQL query, we use 20 RMSs with the resource configuration equal to or even better than the requirement of sub-jobs. Those RMSs have already had some initial workload in their resource reservation profiles and bandwidth reservation profiles. In the experiment, 30% of the number of RMS having CPU performance is equal to the requirement, 60% of the number of RMS having CPU performance is 100% more powerful than the requirement, 10% of the number of RMS having CPU performance is 200% more powerful than the requirement. Along with the increase in performance, the price for each CPU class is also increased. Some important parameters of 20 RMSs are described in Table 3.

More detail about the description of resource configurations and workload configurations can be seen at the address: [http://it.i-u.de/schools/altmann/DangMinh/desc\\_expe2.txt](http://it.i-u.de/schools/altmann/DangMinh/desc_expe2.txt).

The mapping is done by using four different algorithms: L-Map, H-Map, DBC and GA. With the GA algorithm, we increase the number of generations until the algorithm gains the best results. Several experiments have shown that GA needs about 2000 generations. To prove the necessary runtime to achieve the highest quality solutions of the GA algorithm, we present here the experiment results of GA algorithm with 1500 generations. The final result of the experiment is presented in Table 4. The cost and runtime of solutions generated by each algorithm correlative with each workflow are recorded in column Cost and Rt (Runtime) respectively.

The experiment is divided into 3 levels. In the simple level, 8 workflows having a number of sub-jobs in the range 21 - 25 are mapped to 20 RMSs. The results show that most solutions created by L-Map, H-Map, GA-1500 and GA-2000 converge to one value with each workflow. The DBC algorithm had the shortest runtime but found some solutions having higher cost than other algorithms. We can also see that the GA algorithm needs much greater time to find the solutions. This is easy to understand because of the computing intensive checking deadline procedure in each generation. With the H-Map algorithm, the runtime varies a lot because it depends on the number of feasible solutions in the solution space. If the number of feasible solutions is not so large, the

Table 3. RMS configurations

name	num_cpu	cpu_speed	storage	num_exp	cost_cpu	cost_stor	cost_tran	cost_exp
RMS1	256	1000	300000	4	0.05	0.007	0.06	0.15
RMS2	128	1000	200000	3	0.05	0.008	0.05	0.133333
RMS3	256	1000	300000	4	0.05	0.007	0.06	0.15
RMS4	256	1000	300000	4	0.05	0.007	0.06	0.15
RMS5	256	1000	300000	4	0.05	0.007	0.06	0.15
RMS6	64	1000	100000	2	0.05	0.008	0.06	0.166667
RMS7	128	2000	200000	3	0.07	0.008	0.05	0.133333
RMS8	256	2000	300000	4	0.07	0.007	0.06	0.15
RMS9	64	2000	100000	2	0.07	0.008	0.06	0.166667
RMS10	128	2000	200000	3	0.07	0.008	0.05	0.133333
RMS11	128	2000	200000	3	0.07	0.008	0.05	0.133333
RMS12	128	2000	200000	3	0.07	0.008	0.05	0.133333
RMS13	128	2000	200000	3	0.07	0.008	0.05	0.133333
RMS14	128	2000	200000	3	0.07	0.008	0.05	0.133333
RMS15	256	2000	300000	4	0.07	0.007	0.06	0.15
RMS16	256	2000	300000	4	0.07	0.007	0.06	0.15
RMS17	128	2000	200000	3	0.07	0.008	0.05	0.133333
RMS18	256	2000	300000	4	0.07	0.007	0.06	0.15
RMS19	64	3000	100000	2	0.09	0.008	0.06	0.166667
RMS20	128	3000	200000	3	0.09	0.008	0.05	0.133333

Table 4. Performance experiment results

ID	L-Map		H-Map		DBC		GA-1500		GA-2000	
	Rt	Cost	Rt	Cost	Rt	Cost	Rt	Cost	Rt	Cost
Simple level experiment										
1	3	1584.15	24	1584.15	0.5	1584.15	24	1584.15	34	1584.15
2	2	1320.60	23	1320.60	0.5	1320.60	23	1320.60	32	1320.60
3	2	1380.40	24	1380.40	0.5	1386.00	24	1380.40	32	1380.40
4	2	1650.81	16	1650.81	0.5	1650.81	24	1650.81	32	1650.81
5	2	1787.08	13	1787.08	0.5	1798.35	24	1787.08	32	1787.08
6	4	1762.43	24	1762.43	0.5	1770.83	31	1762.43	39	1762.43
7	3	1780.12	23	1780.12	0.5	1782.98	31	1780.12	41	1780.12
8	3	1681.94	32	1681.94	1	1684.74	31	1681.94	41	1681.94
Intermediate level experiment										
9	3	2148.16	13	2153.82	0.5	2159.43	31	2148.16	41	2148.16
10	3	1536.58	32	1536.58	0.5	1536.58	31	1536.58	39	1536.58
11	4	1862.64	54	1862.64	0.5	1873.84	35	1862.71	43	1862.64
12	5	2005.05	37	2005.05	0.5	2005.05	32	2005.05	43	2005.05
13	4	2058.08	44	2058.08	1	2061.07	32	2060.95	46	2058.15
14	4	2239.63	33	2239.63	0.5	2253.63	32	2242.50	43	2242.50
Advance level experiment										
15	5	1778.99	43	1778.99	0.5	1784.59	32	1778.99	46	1778.99
16	9	2179.29	20	2190.49	0.5	2193.23	40	2182.09	55	2179.29
17	9	2184.89	19	2187.75	0.5	2201.63	38	2187.63	54	2187.63
18	6	2539.36	17	2542.35	1	2539.49	38	2539.46	54	2539.42
19	4	2264.72	39	2264.72	0.5	2273.12	38	2264.72	56	2264.72
20	9	1983.74	25	1983.74	0.5	1983.74	41	1983.74	56	1983.74

procedure to form the set of initial solutions which requires the checking of deadlines of many candidate configurations is invoked. Therefore, the runtime is much greater and vice versa. The L-Map algorithm found a solution in just few seconds.

In the intermediate level experiment, we mapped 6 workflows with the number of sub-jobs in the range from 25 to 28. Through the result of this experiment, with the runtime aspect, the situation is similar to the simple level experiment. However, there is a difference in the quality of solutions

found by different methods. The DBC algorithm which does not use local search found lower quality solutions than other methods. Both H-Map and GA-1500 algorithm found one solution having an equal or higher cost than L-Map and GA-2000.

The advanced level experiment maps 6 workflows with the number of sub-jobs in the range from 28 to 32 sub-jobs. There is no great difference in the situation of the runtime aspect compared to the two above experiments. In this experiment, beside the DBC algorithm, we can clearly see the increase in the number of lower quality solutions found by H-Map and GA-1500 compared to L-Map and GA-2000. The result of this experiment shows that the L-Map algorithm found equal or even higher quality solutions than the GA-2000 algorithm.

In the three levels of the experiment, the runtime of the DBC algorithm is the shortest because it is the simplest algorithm and has the lowest quality results. To improve the quality of the mapping solution, we have to apply some specific techniques and thus the runtime is extended by few seconds. From the experiment data, we can see that the L-Map algorithm finds 13 out of 20 higher quality mapping solutions than the DBC algorithm.

The cost difference between solutions found by the L-Map algorithm and the DBC algorithm is small in absolute value. However, when we examine the difference within a business context and a business model, it will have significant meaning. In our business model [6], the broker performs the workflow execution service which includes mapping, executing, monitoring and error recovery. When the workflow execution has finished, it settles the accounts. It pays the service providers and charges the end-user. The profit of the broker is the difference. The value-added that the broker provides is the handling of all the tasks for the end-user. From the business point of view, the broker does the mapping and the income is more important than the total cost of running the workflow. Assuming that the workflow execution service counts for 5% of the total running cost, the broker using the L-map algorithm will have the income 6% higher than the broker using the DBC algorithm.

Moreover, experimenting the negotiation period of the SLA workflow with Web service technology, each negotiation round took a minute or more, mainly for user checking the differences in SLA content. Thus, in our opinion, the runtime of the L-Map algorithm is well acceptable in practical with just few seconds.

## 5 Conclusion

This paper has presented a method, which performs an efficient and precise assignment of light communication workflow to Grid resources with respect to SLAs defined deadlines and cost optimization. In our work, the distinguishing characteristic is that the amount of data to be transferred among sub-jobs is very small. Therefore, we can detect and resolve conflict periods. The performance evaluation showed that the proposed algorithm creates solution of equal or better quality than previous algorithms and needs significantly shorter computation time. Which are decisive factors for the applicability of the method in real environments because large-scale workflows can be planned and assigned efficiently.

## References

- [1] Quan, D.M., and Kao, O., 2005, Mapping Grid job flows to Grid resources within SLA context. Paper presented at the European Grid Conference (EGC 2005), Amsterdam, The Netherlands, 14-16 February.
- [2] Quan, D.M., and Kao, O., 2005, On Architecture for an SLA-aware Job Flows in Grid Environments. *Journal of Interconnection Networks*, 6, 245-264.
- [3] Quan, D.M., 2007, Error recovery mechanism for grid-based workflow within SLA context. *Int. J. High Performance Computing and Networking*, 5, 110-121
- [4] Quan, D.M., 2006, Mapping Heavy Communication Workflows onto Grid Resources Within an SLA Context. Paper presented at the 2th High Performance Computing and Communication Conference, Munich, Germany, 13-15 September.
- [5] Quan, D.M. and Kao, O., 2005, SLA negotiation protocol for grid-based workflows. Paper presented at the International Conference on High Performance Computing and Communications (HPPC-05), Sorrento, Italy, 21-23 September.

- [6] Quan, D.M. and Altmann, J., 2008, Bilateral bargaining game and fuzzy logic in the system handling SLA-based workflow. Paper presented at the Symposium on Web and Mobile Information Services (WAMIS08), Okinawa, Japan, 27 March.
- [7] Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., and Chang, H., 2004, QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*, 30, 311-327.
- [8] Brandic I., Benkner, S., Engelbrecht, G. and Schmidt, R., 2005, QoS Support for Time-Critical Grid Workflow Applications, Paper presented at the e-Science 2005 conference, Melbourne, Australia, 5-8 December.
- [9] 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. Paper presented at the 2nd European Across Grids Conference, Nicosia, Cyprus, 28-30 January.
- [10] Cao, J., Jarvis, S. A., Saini, S. A., and Nudd, G. R., 2003, GridFlow: Workflow Management for Grid Computing. Paper presented at the 3rd IEEE/ACM Int. Symp. on Cluster Computing and the Grid, Tokyo, Japan, 12-15 May.
- [11] Lovas, R., Dzsza, G., Kacsuk, P., Podhorszki, N., and Drtos, D., 2004, Workflow Support for Complex Grid Applications: Integrated and Portal Solutions. Paper presented at the 2nd European Across Grids Conference, Nicosia, Cyprus, 28-30 January.
- [12] Hovestadt, M., Kao, O., Keller, A., Streit, A., 2003, Scheduling in HPC Resource Management Systems: Queuing vs. Planning. Paper presented at the 9th Workshop on JSSPP at GGF8, Washington, USA, 24 June.
- [13] Spooner, D. P., Jarvis, S. A., Cao, J., Saini, S. and Nudd G. R., 2003, Local grid scheduling techniques using performance prediction. Paper presented at the Computers and Digital Techniques 2003 conference, Munich, Germany, 3-7 March.
- [14] Brucker, P., 2004, Scheduling Algorithm, Third edition. Springer Verlag.
- [15] Kumar, V., 1992. Algorithms for constraint-satisfaction problems: a survey. *AI Magazine*, 13, 32-44.
- [16] Sadeh, N. and Fox, M.S., 1996, Variable and value ordering heuristics for the job shop constraint satisfaction problem. *Artificial Intelligence*, 86, 1-41.
- [17] 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. Paper presented at the Principles and Practices of Constraint Programming conference - CP96, Cambridge, USA, 19-22 August.
- [18] Glover, G., 1989, Tabu search: Part I. *ORSA Journal on Computing*, 1, 190-206.
- [19] Glover, F., 1990. Tabu search: Part II. *ORSA Journal on Computing*, 2, 4-32.
- [20] Kirkpatrick, S., Gelatt, C. and Vecchi, M., 1983. Optimization by simulated annealing. *Science*, 220, 671-680.
- [21] 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*, 32, 1-13.
- [22] Nowicki, E., and Smutnicki, C., 2005, An advanced taboo search algorithm for the job shop problem. *Journal of Scheduling*, 8, 145-159.
- [23] Mastrolilli, M., and Gambardella, L.M., 2000, Effective neighborhood functions for the flexible job shop problem. *Journal of Scheduling*, 3, 3-20.
- [24] Jansen K., Mastrolilli, M., and Solis-Oba, R., 2000, Approximation algorithms for flexible job shop problems. Paper presented at the Latin American Theoretical Informatics conference (LATIN'2000), Punta del Este, Uruguay, 10-14 April.
- [25] Gary, M. R. and Johnson, D. S., 1979, Computers and Intractability: A Guide to the theory of NP-Completeness. W. H. Freeman and Co.
- [26] Kohler, W. H. and Steiglitz, K., 1974, Characterization and theoretical comparison of branch-and-bound algorithms for permutation problems. *Journal of ACM*, 21, 140-156.
- [27] Kwok Y. K. and Ahmad, I., 1999, Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)*, 31, 406-471.
- [28] Adam, T. L., Chandy, K. M. and Dickson, J. R., 1974, A comparison of list scheduling for parallel processing systems. *Communication of the ACM*, 17, 685-690.
- [29] Coffman, E. G., 1976, Computer and Job-Shop Scheduling Theory. John Wiley and Sons, Inc., New York, NY.
- [30] Gerasoulis, A. and Yang, T., 1992, A comparison of clustering heuristics for scheduling DAG's on multiprocessors. *J. Parallel and Distributed Computing*, 16, 276-291.
- [31] Sarkar, V., 1989, Partitioning and Scheduling Parallel Programs for Multiprocessors. MIT

- Press, Cambridge, MA.
- [32] Colin, J. Y. and Chretienne, P., 1991, Scheduling with small computation delays and task duplication. *Operation Research*, 39, 680-684.
  - [33] 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.
  - [34] Kruatrachue, B., and Lewis, T., 1988, Grain size determination for parallel processing. *IEEE Software*, 5, 23-32.
  - [35] Sih, G. C., and Lee, E. A., 1993, Declustering: a new multiprocessor scheduling technique. *IEEE Transactions on Parallel and Distributed Systems*, 4, 625-637.
  - [36] Colin, J. Y. and Chretienne, P., 1991, Scheduling with small computation delays and task duplication. *Operation Research*, 39, 680-684.
  - [37] Rewini, H. E. and Lewis, T. G., 1990, Scheduling parallel program tasks onto arbitrary target machines. *Journal of Parallel and Distributed Computing*, 9, 138-153.
  - [38] Shahid, A., Muhammed, S. T. and Sadiq, M., 1994, GSA: scheduling and allocation using genetic algorithm. Paper presented at the Conference on European Design Automation, Paris, France, 19-23 September.
  - [39] Hou, E. S. H., Ansari, N., and Ren, H., 1994, A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 5, 113-120.
  - [40] Buyya, R., Murshed, M., Abramson, D. and Venugopal, S., 2005, Scheduling Parameter Sweep Applications on Global Grids: A Deadline and Budget Constrained Cost-Time Optimisation Algorithm, *Software: Practice and Experience (SPE)*, 35, 491-512.
  - [41] Jia Yu and Rajkumar Buyya, 2006, Scheduling Scientific Workflow Applications with Deadline and Budget Constraints using Genetic Algorithms, *Scientific Programming Journal*, 14, 217-230.
  - [42] Burchard, L., Hovestadt, M., Kao, O., Keller, A. and Linnert, B., 2004, The Virtual Resource Manager: An Architecture for SLA-aware Resource Management, Paper presented at the IEEE CCGrid 2004 conference, Chicago, USA, 19-22 April.