Comput Syst Sci & Eng (2009) 5: 15–28
© 2009 CRL Publishing Ltd

International Journal of
Computer Systems
Science & Engineering

Improving the capability of the SLA workflow broker with parallel processing technology

Dang Minh Quan[1], Jörn Altmann[1] and Laurence T. Yang[2]

[1] *International University in Germany, School of Information Technology, Bruchsal 76646, Germany.*
E-mail: {quandm@upb.de,jorn.altmann@acm.org}
[2] *St. Francis Xavier University, Department of Computer Science, Antigonish, NS, B2G 2W5, Canada*
E-mail: ltyang@stfx.ca

Service Level Agreements (SLAs) are currently one of the major research topics in Grid Computing. Among many system components for supporting of SLA-aware Grid jobs, the SLA mapping module holds an important position and the capability of the mapping module depends on the runtime of the mapping algorithm. With the previously proposed mapping algorithms, the mapping module may develop into the bottleneck of the system if many requests come in during a short period of time. This paper presents parallel mapping algorithms, which can cope with the problem. Performance measurements thereby deliver evaluation results showing the quality of the method.

1. INTRODUCTION

In the Grid Computing environment, many users need the results of their calculations within a specific period of time. Examples of those users are weather forecaster running weather forecasting workflows, automobile producer running dynamic fluid simulation workflow [1]. Those users are willing to pay for getting their work completed on time. However, this requirement must be agreed on by both, the users and the Grid provider, before the application is executed. This agreement is kept in the Service Level Agreement (SLA) [2]. In general, SLAs are defined as an explicit statement of expectations and obligations in a business relationship between service providers and customers. SLAs specify the a-priori negotiated resource requirements, the quality of service (QoS), and costs. The application of such an SLA represents a legally binding contract. This is a mandatory prerequisite for the Next Generation Grids. We presented a prototype system supporting SLAs for the Grid-based workflow in [3, 4, 5, 7].

1.1 Grid-based workflow model

Workflows received enormous attention in the databases and information systems research and development community [9, 10, 11]. According to the definition from the Workflow Management Coalition (WfMC) [12], a workflow is *"The automation of a business process, in whole or parts, where documents, information or tasks are passed from one participant to another to be processed, according to a set of procedural rules."* Although business workflows have great influence on research, another class of workflows emerged in sophisticated scientific problem-solving environments, which is called Grid-based workflows [13, 14, 1]. A Grid-based workflow differs slightly from the WfMC definition as it concentrates on intensive computation and data analyzing but not on the business process. A Grid-based workflow is characterized by the following features [15, 16]:

- A Grid-based workflow usually includes many sub-jobs (i.e. applications), which perform data analysis tasks. However, those sub-jobs are not executed freely but in a strict sequence.

- A sub-job in a Grid-based workflow depends tightly on the output data from previous sub-jobs. With incorrect input data, a sub-job will produce wrong results and damage the result of the whole workflow.
- Sub-jobs in the Grid-based workflow are usually computationally intensive. They can be sequential or parallel programs and require a long runtime.
- Grid-based workflows usually require powerful computing facilities (e.g. super-computers or clusters) to be run on.

Like many popular systems handling Grid-based workflows [17, 18, 1], our system is of the Directed Acyclic Graph (DAG) form. The user specifies the required resources needed to run each sub-job, the data transfer between sub-jobs, the estimated runtime of each sub-job, and the expected runtime of the whole workflow.

In this paper, we assume that time is split into slots. Each slot equals a specific period of real time, from 3 to 5 minutes. We use the time slot concept in order to limit the number of possible start-times and end-times of sub-jobs. More over, delaying 3 minutes also has little meaning with the customer. It is noted that data to be transferred between sub-jobs can be very large.

1.2 Grid service model

The computational Grid includes many High Performance Computing Centers (HPCCs). The resources of each HPCC are managed by a software called local Resource Management System (RMS)[1]. Each RMS has its own unique resource configuration. A resource configuration comprises the number of CPUs, the amount of memory, the storage capacity, the software, the number of experts, and the service price. To ensure that the sub-job can be executed within a dedicated time period, the RMS must support advance resource reservation such as CCS [19]. Figure 1 depicts an example of an CPU reservation profile of such an RMS. In our model, we reserve three main types of resources: CPU, storage, and expert. The addition of further resources is straightforward.

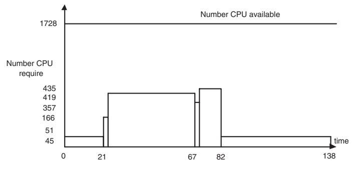


Figure 1 A sample CPU reservation profile of a local RMS

If two output-input-dependent sub-jobs are executed on the same RMS, it is assumed that the time required for the data transfer equals zero. This can be assumed since all compute nodes in a cluster usually use a shared storage system like NFS or DFS. In all other cases, it is assumed that a specific amount of data will be transferred within a specific period of time, requiring the reservation of bandwidth.

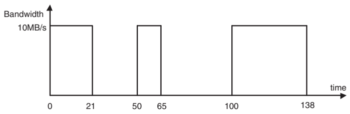


Figure 2 A sample bandwidth reservation profile of a link between two local RMSs

The link capacity between two local RMSs is determined as the averagely available capacity between those two sites in the network. The available capacity is assumed to be different for each different RMS couple. Whenever a data transfer task is required on a link, the possible time period on the link is determined. During that specific time period, the task can use the whole capacity, and all other tasks have to wait. Using this principle, the bandwidth reservation profile of a link will look similar to the one depicted in Figure 2. A more realistic model for bandwidth estimation (than the average capacity) can be found in [20]. Note, the kind of bandwidth estimation model does not have any impact on the working of the overall mechanism.

1.3 Business model

In the case of Grid-based workflow, letting users work directly with resource providers has two main disadvantages:

- The user has to have sophisticated resource discovery and mapping tools in order to find the appropriate resource providers.
- The user has to manage the workflow, ranging from monitoring the running process to handling error events.

To free users from this kind of work, it is necessary to introduce a broker handling the workflow execution for the user. We proposed a business model [6] for the system as depicted in Figure 3. There are three main entities: the end-user, the SLA broker and the service provider:

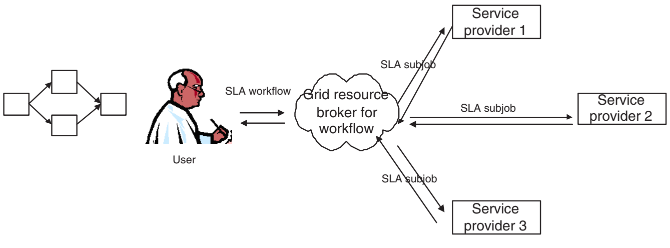


Figure 3 Stakeholders and their business relationship

The end-user wants to run a workflow within a specific period of time. The user asks the broker to execute the workflow for him and pays the broker for the workflow execution service. The user does not need to know in detail how much he has to pay to each service provider. He only needs to know the total amount. This amount depends on the urgency of the workflow and the budget of the user. If there is a SLA violation, for example the runtime deadline

[1] In this paper, RMS is used to represent the cluster/super computer as well as the Grid service provided by the HPCC.

has not been met, the user will ask the broker for compensation. This compensation is clearly defined in the Service Level Objectives (SLOs) of the SLA.

The SLA workflow broker represents the user as specified in the SLA with the user. It controls the workflow execution. This includes mapping of sub-jobs to resources, signing SLAs with the services providers, 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-add that the broker provides is the handling of all the tasks for the end-user.

The service providers execute the sub-jobs of the workflow. In our business model, we assume that each service provider fixes the price for its resources at the time of the SLA negotiation. As the resources of a HPCC usually have the same configuration and quality, each service provider has a fixed policy for compensation if its resources fail. For example, such a policy could be that $n\%$ of the cost will be compensated if the sub-job is delayed one time slot.

Figure 4 depicts a sample scenario of running a workflow on the Grid environment.

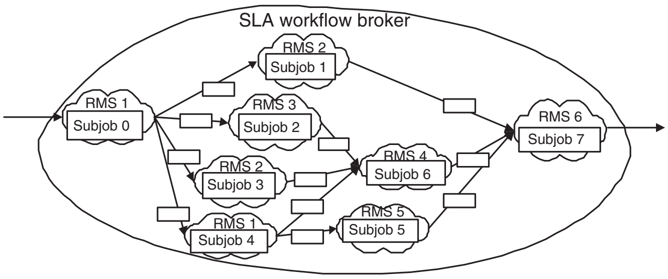


Figure 4 A sample running Grid-based workflow scenario

1.4 Problem statement

The formal specification of the described problem includes following elements:

- Let R be the set of Grid RMSs. This set includes a finite number of RMSs which provide static information about controlled resources and the current reservations/assignments.
- Let S be the set of sub-jobs in a given workflow including all sub-jobs with the current resources and deadline requirements.
- Let E be the set of data transfers in the workflow, which expresses 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 of sub-job s_i. This set includes all RMSs, which can run sub-job s_i, $K_i \subset R$.

Based on the given input, a feasible and possibly optimal solution is sought allowing 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 as Formula 1.

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

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

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

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 dependencies of the sub-jobs are resolved and the execution order remains unchanged.
- **Criterion 4:** The capacity of an RMS must equal or be greater than the requirement at any time slot. Each RMS provides a profile of currently available resources and can run many sub-jobs of a single workflow 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, 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.
- **Criterion 5:** The data transmission task e_{ki} from sub-job s_k to sub-job s_i must not overlap other reserved data transmission tasks on the link between RMS running sub-job s_k to RMS running sub-job s_i. $e_{ki} \in E$.

In the next phase the feasible solution with the lowest cost is sought. The cost C of a Grid workflow is defined by Formulas 3, 4, 5. It is a sum of four factors: money for using the CPU, money for using the storage, cost of using the experts knowledge and finally money for transferring data between the involved resources.

$$C_1 = \sum_{i=1}^{n} s_i.r_t * (s_i.n_c * r_j.p_c + s_i.n_s * r_j.p_s + s_i.n_e * r_j.p_e) \tag{3}$$

$$C_2 = \sum e_{ki}.n_d * r_j.p_d \tag{4}$$

$$C = C_1 + C_2 \tag{5}$$

with $s_i.r_t$, $s_i.n_c$, $s_i.n_s$, $s_i.n_e$ are the runtime, number of CPUs, volume of storage, number of experts of sub-job s_i respectively. $r_j.p_c$, $r_j.p_s$, $r_j.p_e$, $r_j.p_d$ are the prices of using CPU, storage, expert, data transmission of RMS r_j respectively. $e_{ki}.n_d$ is the number of data to be transferred from sub-job s_k to sub-job s_i. It is noted that we do not consider the problem of data caching [29, 30, 31] in this work.

If two sequential sub-jobs run on the same RMS, the cost of transferring data from the previous sub-job to the later sub-job is neglected. It can be easily shown that the optimal mapping of the workflow to Grid RMS with cost optimizing is an NP hard problem.

The ability to find a good solution depends mainly on the resource state at the expected period when the workflow runs. During that period, if the number of free resources in the profile is large, there are a lot of feasible solutions and we can choose the cheapest one. But if the number of free resources in the profile is small, simply finding out a feasible solution is difficult. Thus, a good mapping mechanism should be able to find out an inexpensive solution when there is a wealth of free resources and to be able to uncover a feasible solution when there are few free resources in the Grid.

In the previous works [5, 4, 7], we proposed a mapping mechanism to map Grid-based workflow onto resources within the SLA context. The mapping mechanism includes 3 algorithms.

- The w-Tabu algorithm [7] to map the Grid-based workflow within the SLA context with execution time optimization. This algorithm is used mainly when the Grid resources are busy.
- The L-Map algorithm [4] to map light communication Grid-based workflow within the SLA context, satisfying the deadline and optimizing the cost. With the light communication workflows, the data to be transferred among sub-jobs is very small, usually less than 10MB. 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.
- The H-Map algorithm [5] to map heavy communication Grid-based workflow within the SLA context, satisfying the deadline and optimizing the cost. A workflow is considered as a heavy communication workflow if the data to be transferred among sub-jobs is greater than 10MB. In fact, this amount of data can be very large, usually at GBs scale. In this case, we have to reserve the bandwidth and pay attention to the cost of data transfer.

The result of the extensive experiment shows that the runtime of the w-Tabu algorithm, the H-Map algorithm and the L-Map algorithm is between 1 to 14 seconds depending on the Grid resource and the size of the workflow. Thus, in a critical case, the SLA workflow broker could serve only four users' SLA requests per minute. With a large and crowded Grid, this capability is obviously insufficient and the SLA workflow broker may wind up being the bottleneck of the system. Thus, reducing the runtime of the mapping algorithms while maintaining the quality of the mapping solutions is an essential requirement.

In this paper, we propose a way of parallelizing mapping algorithms based on the proposed algorithms to increase the capability of the SLA workflow broker. The parallel algorithms, called pw-Tabu, pH-Map, and pL-Map, will reduce the time for mapping workflows to Grid resources without decreasing the quality of the solution.

The paper is organized as follows. Section 2 describes related works concerning this problem. Section 3 presents the algorithms. The experiment about the quality and the applicability of the proposed algorithm is discussed in Section 4. Section 5 provides a short summary of the paper.

2. RELATED WORKS

2.1 General literature

Many previous pieces of works focus on optimizing the execution time of the workflow. Among them, the old but well-known algorithm Condor-DAGMan from the work of [21] is still used in some present Grid systems. The algorithm makes local decisions about which job to send to which resource and consider only jobs, which are ready to run at any given instance. Also using dynamic scheduling approach, [22] and [23] apply many techniques to frequently rearrange the workflow and reschedule the workflow in order to reduce the runtime of the workflow. Those methods are not suitable for the context of resource reservation because whenever you cancel a reservation you have to pay a fee. Frequently rescheduling may lead to highly running workflow cost.

In recent works, [24] proposed x-DCP algorithm. The DCP algorithm is based on the principle of continuously shortening the longest path (also called critical path (CP)) in the task graph, by scheduling tasks in the current CP to an earlier start time. [25] and [26] proposed *Min-min*, *Max-min*, *Sufferage* algorithms. The *Min-min* algorithm uses the Minimum MCT (Minimum Completion Time) as a metric, meaning that the task that can be complete the earliest is given priority. The motivation behind *Min-min* is that assigning tasks to hosts that will execute them the fastest will lead to an overall reduced makespan. In the *Max-min* algorithm, the max-min's metric is the Maximum MCT. The expectation is to overlap long-running tasks with short-running ones. In the *Sufferage* algorithm, the rationale behind sufferage is that a host should be assigned to the task that would "suffer" the most if not assigned to that host. For each task, its sufferage value is defined as the difference between its best MCT and its second-best MCT. Tasks with high sufferage value take precedence. [27] proposed the GRASP algorithm. In this approach a number of iterations are made to find the best possible mapping of jobs to resources for a given workflow. On each iteration, an initial allocation is constructed in a greedy phase. All of those algorithms concentrate on optimizing the execution time of the workflow with parameter sweep tasks on Grid resources. They all assume each task as a sequential program and each resource as a compute node. When applying those approaches to our problem, the experiment results show that the quality of solutions found by those algorithms is not sufficient [5].

Our problem has a close relation to the classical job shop scheduling problem (JSSP), the flexible job shop scheduling problem (FJSSP) [28], multiprocessor scheduling precedence-constrained task graph problem [32]. Our problem differs from those problems in many factors. 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.

Optimizing the cost of running the workflow while still meeting the expected deadline has recently received many attentions from the scientists. The mapping of Grid workflows onto Grid resources based on existing planning technology is presented in [17]. 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 insufficient memory [8]. 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 separate works [33, 34], Zeng, et al and Iwona, et al built systems to support QoS features for a Grid-based workflow. In their work, a workflow includes many sub-jobs, which are sequential programs, and a Grid service has ability to handle one sub-job at a time. To map the workflow on to the Grid services, they used the Integer Programming method. But applying Integer Programming to our problem creates many difficulties. The first is the flexibility in runtime of the data transfer task. The time to complete this depends on the bandwidth and the reservation profile of the link and this varies from link to link. The variety in completion time of the data transfer task makes the constraints presentation very complicated. The second is that an RMS can handle many parallel programs at a time. Thus, presenting the constraints of profile resource requirement and profile of resource availability in Integer Programming is very difficult to perform.

The original DBC Grid scheduling algorithm [35], called the cost-time optimization scheduling algorithm, is used to schedule parameter sweep applications on global Grids. The algorithm is built on the cost-optimization and time-optimization scheduling algorithms. This is accomplished by first selecting suitable resources according to the cost priority. After that, the time-optimization algorithm is applied 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. The experimental result shows that this application has insufficient quality [4].

2.2 w-Tabu algorithm

In previous work [7], we proposed the w-Tabu algorithm. The overview of w-Tabu algorithm is presented in Algorithm 1.

Algorithm 1 w-Tabu algorithm

```
Determine assigning sequence for all sub-jobs of the workflow
Generate reference solutions set
for all solutions in reference set do
    Improve the solution as far as possible with the modified Tabu search
end for
Pick the solution with best result
```

The assigning sequence is based on the latest start_time of the sub-job. Sub-jobs having smaller latest start time will be assigned earlier. Each solution in the reference solutions set can be thought of as the starting point for the local search so it should be spread as wide as possible in the searching space. To satisfy the space spread requirement, the number of similar map $sub - job : RMS$ between two solutions, must be as small as possible. The improvement procedure based on Tabu search has some specific techniques to reduce the computation time. More information about w-Tabu algorithm can be seen in [7].

2.3 L-Map algorithm

In [4], we presented an algorithm called L-Map to map light communication workflows onto the Grid resources. The main idea of the L-Map algorithm is that an initial good and feasible solution is created. From this solution, we will reduce the solution space. The reduced solution space will be searched carefully to find out the best solution by using local search. The skeleton of the algorithm is described in Algorithm 2.

Algorithm 2 L-Map algorithm

```
Create and sort the configuration space
{Find the initial feasible solution}
Create the initial configuration
repeat
    Compute earliest-latest timetable of the configuration
    Build reservation profiles of the related RMSs
    if having conflicts in reservation profiles then
        Adjust sub-jobs of the conflict period or move sub-jobs to other RMSs
    end if
until There are no conflicts in reservation profiles
Limit the configuration space
Create the set of initial configurations
for all configurations in the set do
    Improve the solution with local search
end for
Pick the solution with best result
```

2.4 H-Map algorithm

In [5], we presented an algorithm called H-Map for mapping heavy communication workflows onto the Grid resources. The main idea of the H-Map algorithm is that a set of initial solutions distributed over the search space according to cost factor will be further refined to locate the best solution. To form the set of initial solutions, the candidate RMSs of each sub-job are sorted by the order of cost. Then a configuration is formed by selecting an RMS at a ranking level. Each configuration is then checked for feasibility and improved by using a local search. The framework of the algorithm is described in Algorithm 3.

3. PARALLEL MAPPING ALGORITHMS

From the description of three algorithms w-Tabu, L-Map, and H-Map, we can see that all of them use local search to improve

Algorithm 3 H-Map algorithm

```
Sort the solution space according to the computation cost
Clear the initial set of solutions
while not enough solutions do
    Form new configuration by combining 2 cost levels
    Compute timetable to check the feasible
    If feasible, put to the initial set of solutions
end while
for all configurations in the set do
    Do local search with the cost function
end for
Pick the solution with best result
```

the quality of different initial configurations. The experimental result shows that the local search part is the main time consuming part. Thus, the main strategy is parallelizing the local search procedures to have 3 parallel algorithms pw-Tabu, pL-Map and pH-Map. All parallel algorithms are implemented using Message Passing Interface (MPI). In this Section, we will describe the common procedures used by all parallel algorithms, the parallel techniques and the implementations.

3.1 Common procedures

All three algorithms pw-Tabu, pL-Map and pH-Map use some common concepts and procedures and they continue to exist in the parallel mapping algorithms.

3.1.1 Forming the configuration space

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 requirements and the RMS's resource configurations is done by several logic checking conditions in the WHERE clause of the SQL SELECT command. This work will satisfy Criterion 1.

3.1.2 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 $a = a_1, a_2, \ldots, a_n$, $a_i \in K_i$, if each sub-job has m candidate RMSs, we generate $n * (m-1)$ configurations a' as in Figure 5. 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.

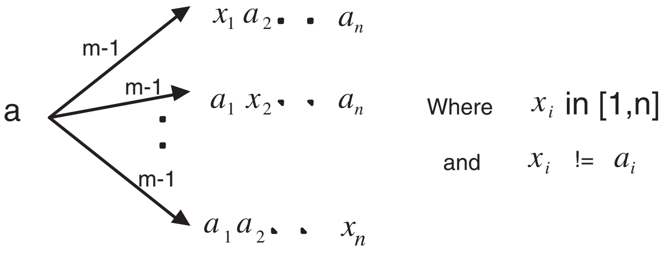


Figure 5 Neighborhood structure of a configuration

3.1.3 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. First, 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. We recognized that the latest time factor is the main parameter to evaluate the full affection of the sequential assigning decision [5]. 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 satisfy Criterion 3.

3.1.4 Determining the timetable of the workflow

To determine the finished time 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 the 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 the state of the RMS's reservation profile. We use the 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 Algorithm 4.

Algorithm 4 Procedure to determine workflow time table

```
for each sub-job $k$ following the assigning sequence do
    for each link from determined sub-jobs to $k$ do
        $min\_st\_tran$=$end\_time$ of source sub-job
        Search reservation profile of the link to have
            $start\_tran > min\_st\_tran$
        $end\_tran = start\_tran + num\_data/bandwidth$
        Store $end\_tran$ in a list
    end for
    $min\_st\_sj$=max ($end\_tran$)
    Search in reservation profile of RMS running $k$ the
        $start\_job > min\_st\_sj$
    $end\_job = start\_job + runtime$
end for
```

For each sub-job of the workflow in the assigning sequence, firstly, we find all the runtime periods of data transfer tasks from previous sub-jobs to the current sub-job. This period must be 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. Then, we determine the runtime period of the sub-job itself. This period must be latter than the latest finish time of previous related data transfer task. The whole procedure is not so complicated but time consuming. The most time consuming steps are located in the searching reservation profiles. This procedure will satisfy the Criteria 4 and 5.

3.2 pw-Tabu algorithm

pw-Tabu algorithm is a parallel algorithm to optimize the execution time of the Grid-based workflow within the SLA context. It is based on the w-Tabu algorithm. The architecture of the algorithm framework is presented in Figure 6.

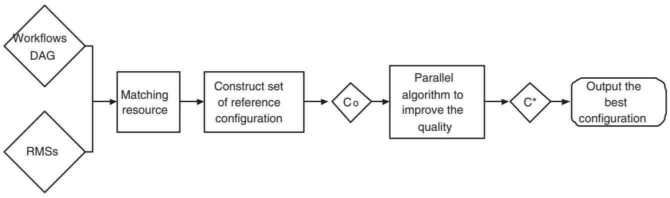

Workflows DAG
RMSs
Matching resource
Construct set of reference configuration
Co
Parallel algorithm to improve the quality
C*
Output the best configuration

Figure 6 The architecture of the pw-Tabu algorithm framework

The inputs of the algorithm are the workflow and the Grid resources. After building the configuration space by matching the sub-job's resource requirements and the RMS's resource configurations, the set C_o of initial solutions is created. Then, a parallel algorithm will improve the quality of each initial solution as far as possible. The best solution is the output.

3.2.1 Matching resources and constructing the set of initial configurations

The procedures of matching resource is described in Section 3.1.1. To construct the reference configurations set, we use the procedure as described in Algorithm 5. Each configuration of the reference configurations set can be thought of as the starting point for a local search so it should be spread as widely as possible throughout the searching space. To satisfy the space spreading requirement, the number of the same map sub-job:RMS between two configurations must be as small as possible. The number of the member in the reference set depends on the number of available RMSs and the number of sub-jobs. During the process of generating a reference solution set, each candidate RMS of a sub-job has a co-relative $assign\_number$ to count the times that RMS is assigned to the sub-job. During the process of building a reference configuration, we use a similar set to store all defined configurations having at least a map sub-job:RMS similar to one in the creating configuration. More detail description about the procedures can be seen in [7].

Algorithm 5 Generating reference set algorithm

```
Set $assign\_number$ of each candidate RMS =0
while $m\_size < max\_size$ do
   Clear the similar set
   for each sub-job in the workflow do
      for each RMS in the candidate list do
         for each solution in the similar set do
            if solution contains sub-job:RMS then
               $num\_sim++$
            end if
            Store tuple $(sub-job, RMS, num\_sim)$ in a list
         end for
      end for
      Sort the list
      Pick the best result
      $assign\_number++$
      if $assign\_number > 1$ then
         Find defined solution having the same sub-job:RMS
         and put to similar set
      end if
   end for
end while
```

3.2.2 Parallel algorithm to improve the quality

The task of improving the quality of the initial configuration set is divided to many sub-tasks. Those sub-tasks are processed in parallel. The algorithm follows the convention master-slave model as depicted in Figure 7.

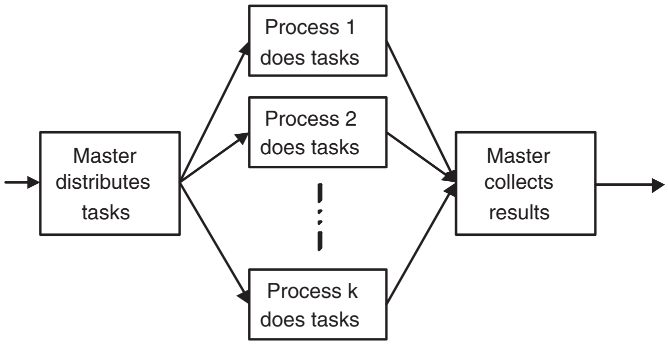


Figure 7 Working model of the improving quality algorithm

To improve the quality of solutions in the initial set C_o, the master process distributes evenly configurations in the set C_o to slave processes. The data sending from the master process to the slave process is presented in Figure 8.

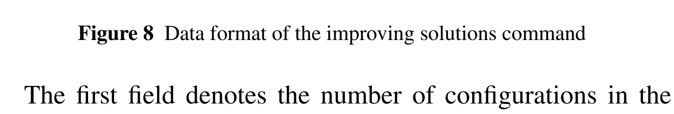
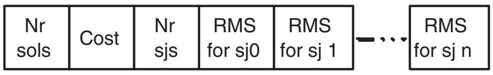


Figure 8 Data format of the improving solutions command

The first field denotes the number of configurations in the message. Each configuration has its number of sub-jobs and list of RMSs for sub-jobs in the workflow.

Each slave process improves the quality of each initial configuration by using local search. The procedure tries to replace the present RMS with other RMSs in the candidate list to find the best improvement. To reduce the computing time, we focus on the critical path of the workflow. The process continues until the solution cannot be improved any more. Detail description about the procedure is presented in [7].

When the improvement is finished, each slave process sends the master only the best found solution with the message presented in Figure 9. The first field always has value of 1 as the slave process always finds out a solution. The cost field denotes the execution time of the solution. The rest fields present the solution with its number of sub-jobs and list of RMSs for sub-jobs in the workflow.



Nr sols	Cost	Nr sjs	RMS for sj0	RMS for sj 1	...	RMS for sj n

Figure 9 Data format of the replying messages

The master picks the best solution, computes the timetable and outputs the result.

3.2.3 Algorithm implementation

The implementation of master and slave process is presented in Algorithm 6 and Algorithm 7.

Algorithm 6 Master process of the pw-Tabu algorithm

```
Get information of workflow and Grid resources
Create solution space
Create the reference set of configurations
Distribute the task of improving initial configurations to slave processes
Collect the improved solutions
Pick the best solution and compute the timetable
Send kill signal to slave process
```

Algorithm 7 Slave process of the pw-Tabu algorithm

```
Get information of workflow and Grid resources
Create solution space
When receive the task of improving initial solution then do it and send back result to the master
When receive the kill signal then exit
```

3.3 pL-Map algorithm

In this Section, we describe a parallel algorithm based on the L-Map algorithm called pL-Map to map light communication workflow onto Grid resources. The architecture of the algorithm framework is presented in Figure 10.

The inputs of the algorithm are the workflow and the Grid resources. After building the configuration space by matching the sub-job's resource requirements and the RMS's resource configurations, we find the initial feasible solutions. Then, a parallel algorithm will improve the quality of the initial solution as far as possible. The best solution is the output.

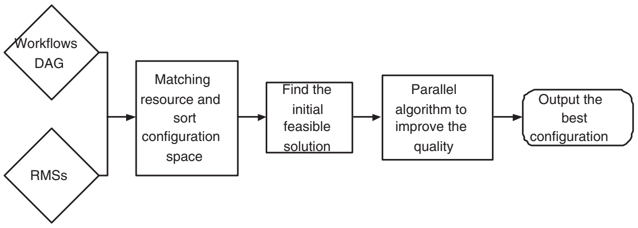

Workflows DAG
RMSs
Matching resource and sort configuration space
Find the initial feasible solution
Parallel algorithm to improve the quality
Output the best configuration

Figure 10 The architecture of the algorithm framework

3.3.1 Finding the initial feasible solution

After building the configuration space by performing the matching resources as described in Section 3.1.1, we sort the candidate RMS list of each sub-job according to the cost order. The procedure of creating the initial solution is similar to the one in the L-Map algorithm. The initial configuration is created by assigning each sub-job to the lowest cost RMS in the candidate list. The initial configuration is adjusted to become the feasible solution by resolving the conflicts in RMSs' resource reservation profile. Detail description about the procedures can be seen in [4].

3.3.2 Parallel algorithm to improve the quality

The task of improving the quality of the initial configuration set is divided to many sub-tasks. Those sub-tasks are processed in parallel. The algorithm follows the convention master-slave model as depicted in Figure 7.

The master process

After having the initial feasible solution, the master module does following steps:

Step 1: Limiting the configuration space. Suppose that each sub-job has m candidate RMSs. Suppose that in the initial 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. We limit the solution space in this way because the higher ranking RMSs only produce out higher cost solution than the initial feasible solution.

Step 2: Creating the initial configurations set. 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.

Step 3: Distributing initial configurations to slave modules. To improve the quality of the initial configuration set, the master process distributes evenly configurations in the set to slave processes. The data sending from the master process to the slave process is presented in Figure 8. The first configuration is the initial feasible solution. We have to include the initial feasible solution to all messages because it is used to limit the search space in each slave process.

Step 4: Selecting the best solution and output. The master process collects all results from the slave processes and outputs the best solution.

The slave process

Each slave module also uses the initial feasible solution to reduce the configuration space as the master did.

After that, each slave process improves the quality of each initial configuration by using local search. The procedure tries to replace the present RMS with other RMSs in the candidate list to find the best feasible improvement. The process continues until the solution cannot be improved any more. Detail description about the procedure is presented in [4].

When the improvement is finished, each slave process sends the master only the best found solution with the message presented in Figure 9. It is noted that the slave process may not find out a feasible solution. In this case, the field “Nr sols” has value zero and the master will ignore the message. Otherwise, it has the value of 1 and the cost field denotes the cost of running the workflow.

3.3.3 Algorithm implementation

The implementation of master and slave process is presented in Algorithm 8 and Algorithm 9 respectively.

Algorithm 8 Master process of the pL-Map algorithm

```
Get information of workflow and Grid resources
Create sorted configuration space
Create the initial feasible solution
Limit the configuration space
Create a set of initial configuration
Distribute the task of improving initial configurations to slave processes
Collect the improved solutions
Pick the best solution and compute the timetable
Send kill signal to slave process
```

Algorithm 9 Slave process of the pL-Map algorithm

```
Get information of workflow and Grid resources
Create sorted configuration space
if Receive the task of improving initial configuration then
    Reduce the configuration space
    for all Received configurations do
        Improve the quality with local search
    end for
    Send back the master the best feasible solution
end if
if Receive the kill signal then
    Exit
end if
```

3.4 pH-Map algorithm

The pH-Map algorithm is a parallel version of the H-Map algorithm to map heavy communication workflow onto the Grid resources. The architecture of the algorithm framework is presented in Figure 11.

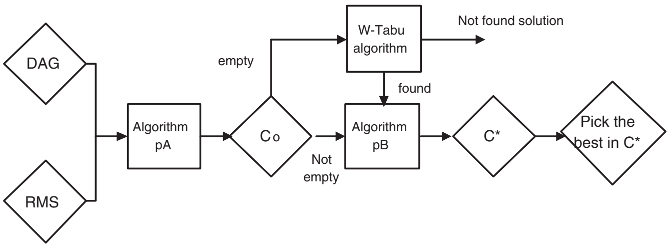


Figure 11 The architecture of the algorithm frame work

The inputs of the algorithm are the workflow and the Grid resources. After building the configuration space by matching the sub-job’s resource requirements and the RMS’s resource configurations, the parallel algorithm pA is invoked to build the set C_o of initial solutions. If C_o is empty, the pw-Tabu algorithm to find the optimal workflow execution time is invoked. Otherwise, the parallel algorithm pB will improve the quality of each initial solution as much as possible. The best solution becomes the output.

3.4.1 Building the sorted configuration space

With each sub-job s_i, we sort the RMSs in the candidate set K_i according to the cost needed to run s_i. The cost is computed according to Formula 5. The configuration space of the sample now can be presented in Figure 12 and Table 1. In Figure 12, the RMSs lying along the axis of each sub-job have increasing cost moving from inside to outside. The line connecting each point in every sub-job axis forms a configuration. Figure 12, for example, presents 3 configurations with an increasing index from inside to outside. Figure 12 also presents the cost distribution of the configuration space according to Formula 5. The configuration in the outer layer has a greater cost than the configuration in the inner layer. The cost of the configuration lying between two layers is greater than the cost of the inner layer and smaller than the cost of the outer layer.

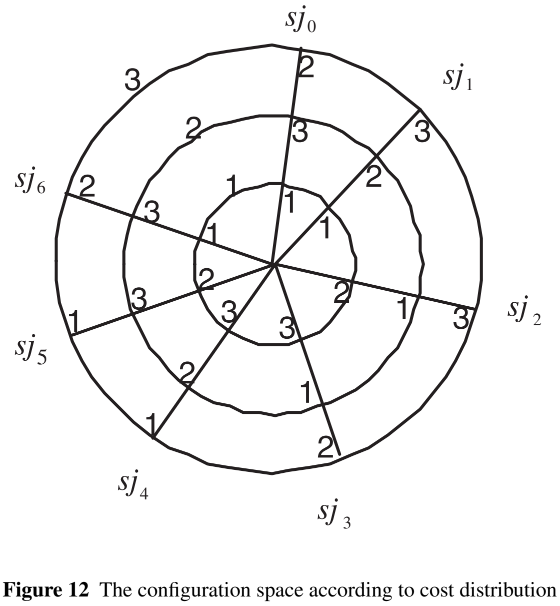


Figure 12 The configuration space according to cost distribution

Table 1 RMS 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

3.4.2 Algorithm pA - Constructing initial solutions

The purpose of algorithm pA is to create a set of initial solutions which is distributed widely over the search space. A configuration can be created in two ways.

- By employing each layer of the sorted configuration space as the configuration.
- By merging two neighborhood layers of the sorted configuration space to form the configuration.

Assume that each sub-job has m candidate RMSs. This means we have m layers in the sorted configuration space and have in total $2*(m-1)$ configurations. With this method we can ensure that the set of initial solutions is distributed over the search space according to cost criteria.

The task of finding the initial solution set is divided into many sub-tasks which are processed in parallel. The algorithm follows the conventional master-slave model as depicted in Figure 7.

The master process

Instead of directly creating the configurations, the master process evenly sends the information about the sorted layers to each slave process so that it can start finding the initial solutions. The format of the data sending from the master process to the slave processes is presented in Figure 13.

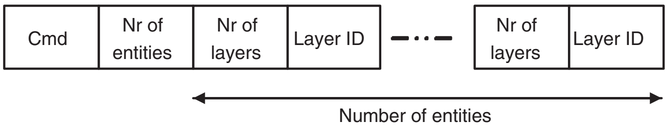


Figure 13 Data format of the finding initial solution command

The first field Cmd is the command from the master to the slaves. The second field, Nr of entities, stores the total number of entities contained in the message. Each entity includes 2 fields. The Nr of layers presents the amount of layers needed to create the configuration. The Layer ID is the index of the layer in the sorted configuration space.

The slave process

If a slave process receives a message with Nr of layer equal to 1, it will create the configuration by employing the layer having its index denoted in the Layer ID.

If a slave process receives a message with Nr of layer equal to 2, it will create the configuration by merging the layer having its index denoted in the field Layer ID and the next layer. For example, if Layer ID equals 1, layer 1 and layer 2 will be merged to produce a new configuration. To do this, we take the p first elements from the first layer and then the p second elements from the second layer and repeat until having enough n elements to form the completed configuration. n is the number of sub-jobs of the workflow. Thus, we get half the number of elements from the first layer and the other half number of elements from the second one. Combining in this way will ensure the target configuration having maximal difference in cost according to Formula 5 comparing to the source configurations.

After receiving a configuration, the slave process checks the Criterion 2 of the configurations. To verify Criterion 2, we have to determine the timetable for all sub-jobs of the workflow. Details about the procedure to perform this task are described in [5]. Here, the procedure is only presented briefly. First, we have to determine the assigning sequence of sub-jobs in the workflow. Then, we determine the most suitable time schedule for each sub-job and data transfer following the assigning sequence.

With entities having Nr of layer equal to 2, if the created solution does not satisfy the Criterion 2 requirement, we construct more to have enough $2*m-1$ configurations. To do the construction, we change the value of p parameter in the range from 1 to $(n/2)$ in order to create the new configuration.

When the slave process finishes computing, it sends the result back to the master. The data sent from the slave process to the master process is presented in Figure 8.

3.4.3 Algorithm pB - Improve the quality of solutions

To improve the quality of solutions in the initial set C_o, the master process evenly distributes solutions in the set C_o to slave processes. We have to collect the initial solutions before redistributing them to the slave processes because each slave process may reveal a different number of solutions. Thus, without the redistribution, the workload in each slave process is not equal.

The data sending from the master process to the slave process is presented in Figure 14. It is similar to the one presented in Figure 8 except that there is one extra field to denote the command type.

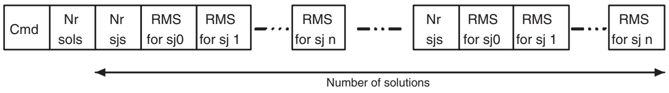


Figure 14 Data format of the improving solutions command

Each slave process improves the quality of each initial solution by using local search. This procedure tries to replace the present RMS with other RMSs in the candidate list to find the best improvement. The process continues until the solution cannot be improved any more. A detailed description about the procedure is presented in [5].

When the improvement is finished, each slave process sends the master only the best found solution with a message similar to the one in Figure 9. The first field always has value of 1 as the slave process received feasible solution from the master, thus, it always finds out at least a feasible solution. The cost field denotes the cost of the solution.

The master then picks the best solution, computes the time tables and outputs it.

3.4.4 Algorithm implementation

The implementation of the master and the slave process is presented in Algorithm 10 and Algorithm 11.

Algorithm 10 Master process of the pH-Map algorithm

```
Get information of the workflow and Grid resources
Create a sorted solution space
Distribute the task of finding the initial solution to the slave processes
Collect initial solutions from slaves
Distribute the task of improving initial solutions to slave processes
Collect the improved solutions
Pick the best solution
Send the kill signal to the slave processes
```

Algorithm 11 Slave process of the pH-Map algorithm

```
Get information of the workflow and Grid resources
Create a sorted solution space
When receiving the task of finding the initial solution then do it and send back the result to the master
When receiving the task of improving the initial solution then do it and send back the result to the master
When receiving the kill signal, exit
```

3.5 Comments

From the described algorithm architectures and implementations of pw-Tabu, pL-Map and pH-Map algorithms, we have the following comments.

- The main strategy of the parallel algorithms still remains as the sequential algorithms. Thus, the quality of the algorithms is kept. Only the computation intensive parts are parallelized in order to improve the execution time of the mapping module.
- As the size of the initial solution set is limited, the scalability of the pH-Map algorithm is also limited. In particular, the maximum effective number of computing nodes equals to the number of initial solutions.
- All master and slave processes have complete information about the workflow and the resources. Thus, the data transfer among processes is reduced.

4. PERFORMANCE EXPERIMENT AND APPLICABILITY

As the quality of the algorithm is unchanged, the performance experiment is simulated with simulation to check for the runtime of the mapping algorithms. The software used in the experiments is rather standard and simple (Linux Ubuntu 7.0, MySQL). The whole simulation program is implemented in C/C++. The hardware for the experiment is a cluster including 8 computing nodes 3,0 Ghz, 1GB memory. 8 computing nodes are connected through switch 100 Mbps.

The goal of the experiment is to measure the time needed for the computation. To do this, 18 workflows with different topologies, number of sub-jobs, sub-job specifications, and amount of data transferring were generated as workload. For the pw-Tabu algorithm, the number of data transferring is in a wide range from very small to very great [7]. For the pL-Map algorithm, the number of date transferring is very small, usually less than 10 Mega Bytes [4]. For the pH-Map algorithm, the number of data transferring is very great, usually in Giga Bytes scale [5]. The Grid resources include 20 RMSs with different resource configurations and different resource reservation contexts.

In the first experiment, we studied the runtime of the algorithms for 18 single workflows with different number of computing nodes. Each computing node runs one slave process. In the case of 1 computing node, we used the sequential algorithm. With more than 1 computing nodes, we used the correlated parallel algorithm. As the time entity in our experiment is in second, the smallest runtime of the algorithm is 1 second. The experimental results of pw-Tabu, pL-Map and pH-Map are presented in Table 2, 3, 4 respectively. The first column presents the index of the workflow. The size of the workflow increases along the increase of the index. Other columns except the final one present the runtime of the mapping algorithms according to the different number of computing nodes. The final column presents the runtime ratio 1CPU/8CPUs.

Table 2 Performance evaluation result of the pw-Tabu algorithm

ID	1 CPU	2	3	4	5	6	7	8	1CPU/8CPUs
Simple level experiment									
1	1s	1s	1s	1s	1s	1s	1s	1s	100%
2	1s	1s	1s	1s	1s	1s	1s	1s	100%
3	1s	1s	1s	1s	1s	1s	1s	1s	100%
4	2s	2s	1s	1s	1s	1s	1s	1s	200%
5	2s	2s	2s	2s	2s	1s	1s	1s	200%
6	3s	2s	2s	2s	2s	1s	1s	1s	300%
7	2s	2s	2s	2s	2s	1s	1s	1s	200%
Intermediate level experiment									
8	2s	2s	2s	2s	2s	2s	2s	1s	200%
9	3s	3s	3s	2s	1s	2s	2s	1s	300%
10	4s	4s	2s	2s	2s	2s	2s	1s	400%
11	4s	4s	3s	3s	2s	2s	2s	1s	400%
12	5s	5s	4s	4s	2s	2s	2s	1s	500%
13	6s	6s	4s	4s	2s	2s	2s	2s	300%
Advance level experiment									
14	5s	5s	3s	4s	3s	2s	2s	2s	250%
15	5s	5s	4s	4s	3s	2s	2s	2s	250%
16	10s	9s	7s	7s	5s	4s	4s	3s	330%
17	12s	11s	9s	8s	6s	5s	4s	3s	400%
18	15s	14s	12s	10s	9s	7s	5s	4s	375%

From the data in Table 2, 3, 4 we can see that the increase in performance of the pw-Tabu, pL-Map and pH-Map algorithm with small workflows is not as clear as with large workflows.

Table 3 Performance evaluation result of the pL-Map

ID	1 CPU	2	3	4	5	6	7	8	1CPU/8CPUs
Simple level experiment									
1	3s	3s	2s	2s	1s	1s	1s	1s	300%
2	3s	2s	2s	2s	1s	1s	1s	1s	300%
3	2s	2s	2s	2s	1s	1s	1s	1s	200%
4	3s	3s	2s	2s	1s	1s	1s	1s	300%
5	4s	4s	3s	2s	2s	1s	1s	1s	400%
6	4s	3s	3s	2s	2s	2s	1s	1s	400%
7	4s	4s	3s	3s	2s	2s	1s	1s	400%
Intermediate level experiment									
8	3s	3s	2s	2s	2s	1s	1s	1s	300%
9	3s	3s	2s	2s	2s	2s	1s	1s	300%
10	3s	3s	2s	2s	2s	1s	1s	1s	300%
11	4s	3s	3s	2s	2s	2s	2s	2s	200%
12	5s	4s	4s	3s	3s	2s	1s	1s	500%
13	4s	3s	3s	2s	2s	2s	1s	1s	400%
Advance level experiment									
14	4s	4s	3s	2s	2s	2s	1s	1s	400%
15	5s	4s	4s	3s	3s	2s	2s	2s	250%
16	9s	7s	6s	6s	5s	4s	3s	3s	300%
17	10s	9s	7s	6s	4s	4s	3s	2s	500%
18	6s	5s	4s	4s	3s	3s	2s	2s	300%

Table 4 Performance evaluation result of the pH-Map algorithm

ID	1CPU	2	3	4	5	6	7	8	1CPU/8CPUs
Simple level experiment									
1	2s	1s	1s	1s	1s	1s	1s	1s	200%
2	2s	2s	1s	1s	1s	1s	1s	1s	200%
3	2s	2s	2s	1s	1s	1s	1s	1s	200%
4	3s	3s	2s	2s	1s	1s	1s	1s	300%
5	3s	3s	2s	1s	1s	1s	1s	1s	300%
6	3s	3s	2s	1s	1s	1s	1s	1s	300%
7	3s	3s	3s	2s	2s	1s	1s	1s	300%
Intermediate level experiment									
8	4s	4s	3s	3s	2s	1s	1s	1s	400%
9	4s	4s	3s	3s	2s	1s	1s	1s	400%
10	5s	5s	3s	3s	2s	2s	1s	1s	500%
11	5s	4s	5s	3s	2s	2s	1s	1s	500%
12	6s	6s	4s	3s	2s	2s	1s	1s	600%
13	7s	6s	4s	4s	2s	2s	2s	1s	700%
Advance level experiment									
14	7s	6s	5s	4s	3s	3s	2s	1s	700%
15	6s	6s	4s	3s	2s	2s	2s	2s	300%
16	8s	7s	5s	4s	4s	3s	2s	2s	400%
17	10s	9s	7s	5s	4s	4s	3s	2s	500%
18	14s	12s	8s	7s	5s	4s	4s	3s	466%

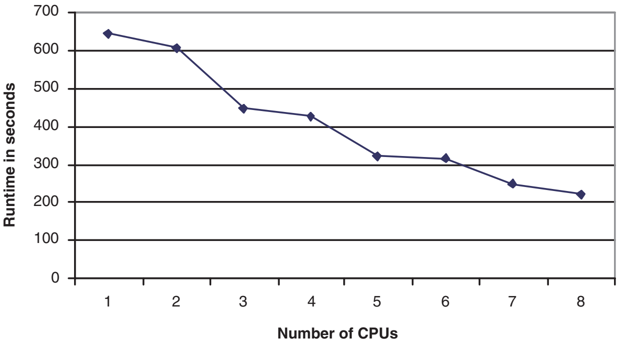


Figure 15 Overall performance of the pw-Tabu algorithm

One reason is that the 1 second resolution is not fine enough for small workflows which already needs short runtime of the H-Map algorithm. Another reason is that the rate between the overhead and the main computing part of the algorithm with small workflows is larger than with the large workflows. Thus, the parallel part applying to small workflows results in a lesser effect.

With the large workflow such as in the advance level experiment, the character of the parallel algorithms can be seen more clearly. The runtime of the algorithms is not reduced linearly with the increasing computing nodes. It is mainly caused by the overhead of parallel processes. The overhead includes the time to load application and input file from a sharing network device, the time to get data from a sharing MySQL database and the time to initialize the data structures. The communication among parallel processes does not have a significant contribution to this delay as there are only one input communication and one output communication per process. When the number of parallel process increases, the overhead increases. Thus, it reduces the acceleration of the algorithm.

To study more carefully the performance of the parallel algorithms, we performed the second experiment with the mixed workload. For this, with each parallel algorithm, we generated 100 requests. Each request was selected randomly from 18 workflows. Then, we continuously mapped the 100 requests with a different number of computing nodes and recorded the needed time to finish the mapping. For the case of one computing node, we used the sequential algorithm. With more than one computing nodes, we used the correlated parallel algorithm. The experimental results of pw-Tabu, pL-Map and pH-Map are presented in Figure 15, 16, 17 respectively.

From Figure 15, 16, 17, we can see the same trend as the above experiment that the speedup of the algorithm is reduced when the number of computing nodes increases. We can also see that the runtime of the algorithm has minor change when the increasing in number of computing nodes is not enough. It is because the workload of the heaviest computing nodes is unchanged. For example, with the pw-Tabu algorithm, with the case of 5 and 6 CPUs in the experiment, the total number of initial solutions is 25 and thus, the heaviest process in both cases must handle 5 initial solutions.

As can be seen in Figure 15, 16, 17, the acceleration of the parallel algorithm is from 300% to 350% with 8 CPUs comparing to 1 CPU. This means that the brokers can serve the users at least 300% faster with 8 CPUs comparing to 1 CPU. Beside that, with the business Grid, the broker could easily have flexible computing power. He could hire many computing nodes in the critical period and return them when the Grid is not crowd. Comparing to the income of the broker, the cost of hiring more computers for mapping is very small. For example, with Amazon pricing schema (13-3-2008), a computing node costs 0,10 $ per hour. Thus, hiring 8 CPUs in 1 hour costs only 0,8 $. This means that the applicability of the approach is very high. By

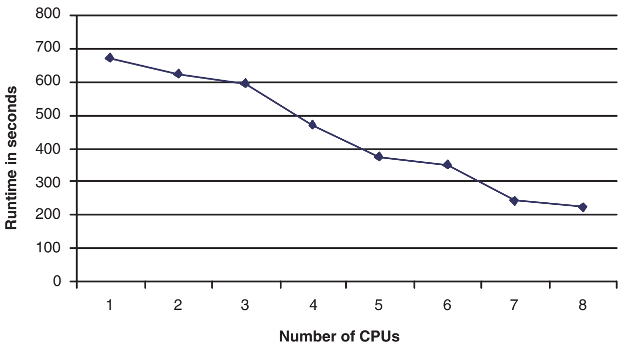


Figure 16 Overall performance of the pL-Map algorithm

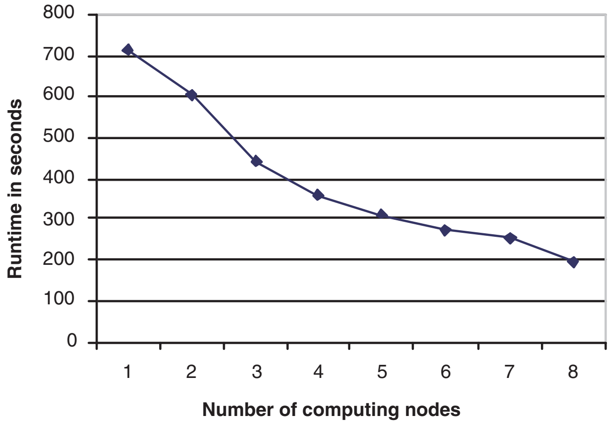


Figure 17 Overall performance of the pH-Map algorithm

applying parallel processing technology, the broker can increase significantly its ability to serve users with low cost.

5. CONCLUSION

This paper has presented a method, which reduces the necessary time to map a Grid-based workflow onto resources within the SLAs context. In particular, we proposed a parallel algorithm pw-Tabu based on the w-Tabu algorithm to optimize the execution time of the workflow, parallel algorithms pL-Map based on the L-Map algorithm and pH-Map based on the H-Map algorithm to map a light and heavy communication workflow respectively onto Grid resources with respect to SLAs defined deadlines and cost optimization. The main strategy of the sequential algorithms still remains while the computing intensive parts are parallelized. Thus, the quality of the algorithm is kept while the runtime is significantly reduced. The performance evaluation showed that the algorithm is very effective especially with large size workflows requiring great computation power. On average, the algorithm can accelerate up to 300% with 8 CPUs. With low cost of hiring computing resources, the method can be applied to real environments without difficulty.

REFERENCES

1. R. Lovas, G. Dózsa, P. Kacsuk, N. Podhorszki, D. Drótos, Workflow Support for Complex Grid Applications: Integrated and Portal Solutions, *Proceedings of 2nd European Across Grids Conference*, Springer LNCS, 2004, pp. 129–138.
2. A. Sahai, S. Graupner, V. Machiraju, and A. Moorsel, Specifying and Monitoring Guarantees in Commercial Grids through SLA, *Proceeding of the 3rd IEEE/ACM CCGrid2003*, 2003, pp. 292–300.
3. D.M. Quan, O. Kao, On Architecture for an SLA-aware Job Flows in Grid Environments, *Journal of Interconnection Networks*, Vol. 6, No. 3, 2005, pp. 245–264.
4. D.M. Quan, J. Altmann, Resource allocation algorithm for light communication Grid-based workflows within an SLA context, Accepted by the International Journal of Parallel, Emergent and Distributed Systems, 2008
5. D.M. Quan, Mapping heavy communication workflows onto grid resources within sla context, *Proceedings of the International Conference of High Performance Computing and Communication (HPCC06)*, 2006, pp. 727–736.
6. D.M. Quan, J. Altmann, Business Model and the Policy of Mapping Light Communication Grid-Based Workflow Within the SLA Context, *Proceedings of the International Conference of High Performance Computing and Communication (HPCC07)*, 2007, pp. 285–295.
7. D.M. Quan, Error recovery mechanism for grid-based workflow within SLA context, *International Journal of High Performance Computing and Networking*, Vol. 5, No. 1/2, 2007, pp. 110–121.
8. D.M. Quan, O. Kao, Mapping Grid job flows to Grid resources within SLA context, *Proceedings of the European Grid Conference,(EGC 2005)*, Springer LNCS press, 2005, pp. 1107–1116.
9. A.K. Elmagarmid, *Database Transaction Models for Advanced Applications*, Morgan Kaufmann, 1992.
10. D. Georgakopoulos, M. Hornick, and A. Sheth, An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure, *Distributed and Parallel Databases*, Vol. 3, No. 2, 1995, pp. 119–153.
11. M. Hsu, (ed.), *Special Issue on Workflow and Extended Transaction Systems*, IEEE Data Engineering, Vol. 16, No. 2, 1993.
12. L. Fischer, *Workflow Handbook 2004*, Future Strategies Inc., Lighthouse Point, FL, USA, 2004.
13. S. Ludtke, P. Baldwin, and W. Chiu, EMAN: Semiautomated Software for High-Resolution Single-Particle Reconstruction, *Journal of Structure Biology*, Vol. 128, 1999, pp. 146–157.
14. G. B. Berriman, J. C. Good, A. C. Laity, Montage: a Grid Enabled Image Mosaic Service for the National Virtual Observatory, *ADASS*, Vol. 13, 2003, pp. 145–167.
15. M. P. Singh, and M. A. Vouk, *Scientific Workflows: Scientific Computing Meets Transactional Workflows*, http://www.csc.ncsu.edu/faculty/mpsingh/ papers/databases/work flows/sciworkflows.html, 1997.
16. J. Yu, R. Buyya, A taxonomy of scientific workflow systems for grid computing', *SIGMOD Record*, Vol. 34, No. 3, 2005, pp. 44–49.
17. E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. Su, K. Vahi, and M. Livny, Pegasus : Mapping Scientific Workflows onto the Grid, *Proceedings of the 2nd European Across Grids Conference*, 2004, pp. 11–20.
18. D. P. Spooner, S. A. Jarvis, J. Cao, S. Saini, and G. R. Nudd, Local Grid Scheduling Techniques Using Performance Prediction, ÎEEE Proceedings - Computers and Digital Techniques, 2003, pp. 87–96.
19. M. Hovestadt, Scheduling in HPC Resource Management Systems:Queuing vs. Planning, *Proceedings of the 9th Workshop on JSSPP at GGF8*, LNCS, 2003, pp. 1–20.

20. R. Wolski, Experiences with Predicting Resource Performance On-line in Computational Grid Settings, *ACM SIGMETRICS Performance Evaluation Review*, Vol. 30, No. 4, 2003, pp. 41–49.
21. CondorVersion 6.4.7 Manual. www.cs.wisc.edu/condor/manual/v6.4 [10 December 2004].
22. R. Duan, R. Prodan, T. Fahringer, Run-time Optimization for Grid Workflow Applications, *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing (Grid'06)*, 2006, pp. 33–40.
23. S. Ayyub, and D. Abramson, GridRod - A Service Oriented Dynamic Runtime Scheduler for Grid Workflows, *Proceedings of the 21st ACM International Conference on Supercomputing*, 2007, pp. 43–52.
24. T. Ma, and R. Buyya, Critical-Path and Priority based Algorithms for Scheduling Workflows with Parameter Sweep Tasks on Global Grids, *Proceeding of the 17th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2005), IEEE CS Press*, 2005, pp. 251–258.
25. Berman *et al.*, New Grid Scheduling and Rescheduling Methods in the GrADS Project, *International Journal of Parallel Programming*, Vol. 33, 2005, pp. 209–229.
26. H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, Heuristics for Scheduling Parameter Sweep applications in Grid environments, *Proceeding of the 9th HeterogeneousComputing workshop (HCW'2000)*, 2000, pp. 292–300.
27. Blythe *et al.*, Task Scheduling Strategies for Workflow-based Applications in Grids, *Proceeding of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)*, 2005, pp. 759–767.
28. P. Brucker, *Scheduling Algorithm*, Third edition, Springer Verlag, 2004.
29. K. Li and H. Shen, Coordinated Enroute Multimedia Object Caching in Transcoding Proxies for Tree Networks. ACM Trans. on Multimedia Computing, Communications and Applications (TOMCAPP), Vol. 5, No. 3, 2005, pp. 289–314.
30. K. Li, H. Shen, F. Y. L. Chin, and W. Zhang, Multimedia Object Placement for Transparent Data Replication, *IEEE Trans. on Parallel and Distributed Systems*, Vol. 18, No. 2, 2007, pp. 212–224.
31. K. Li, H. Shen, F. Y. L. Chin, and S. Q. Zheng, 2005, Optimal Methods for Coordinated Enroute Web Caching for Tree Networks. ACM Trans. on Internet Technology (TOIT), Vol. 5, No. 3, pp. 480–507.
32. M. R. Gary, and D. S. Johnson, (1979), *Computers and Intractability: A Guide to the theory of NP-Completeness*. W. H. Freeman and Co.
33. L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, H. Chang, "QoS-Aware Middleware for Web Services Composition", *IEEE Transactions on Software Engineering*, IEEE CS press, Vol. 30, No. 5, 2004, pp. 311–327.
34. I. Brandic, S. Benkner, G. Engelbrecht, and R. Schmidt, QoS Support for Time-Critical Grid Workflow Applications, *Proceedings of the first International Conference on e-Science and Grid Computing*, IEEE CS press, 2005, pp. 108–115.
35. R. Buyya, M. Murshed, D. Abramson, and S. Venugopal, Scheduling Parameter Sweep Applications on Global Grids: A Deadline and Budget Constrained Cost-Time Optimisation Algorithm, *Software: Practice and Experience (SPE)*, 35, 2005, pp. 491–512.