Error recovery mechanism for grid-based workflow within SLA context

Dang Minh Quan

Paderborn Center for Parallel Computing (PC^2),
University of Paderborn, Fuerstenalle 11,
Paderborn, 33102, Germany
E-mail: quandm@upb.de
Website: <www.upb.de/pc2>

Abstract: Service Level Agreements (SLAs) serve as a foundation for a reliable and predictable job execution at remote grid sites. In this paper, we describe an error recovery mechanism for workflow within the SLA context, coping with catastrophic failure when one or several High Performance Computing Centers (HPCCs) are detached from the grid system. We propose an algorithm to detect all affected sub-jobs when the error happens and an algorithm to remap those sub-jobs to the remaining healthy HPCCs with makespan optimise. The experiment result shows that our mechanism discovers a higher quality solution in a shorter time period than other existing methods.

Keywords: grid computing; service level agreement; SLA; error recovery; workflow; mapping.

Reference to this paper should be made as follows: Quan, D.M. (2007) 'Error recovery mechanism for grid-based workflow within SLA context', *Int. J. High Performance Computing and Networking*, Vol. 5, Nos. 1/2, pp.110–121.

Biographical notes: Dang Minh Quan has been a researcher at Paderborn Center for Parallel Computing (PC^2), University of Paderborn, Germany, since 2004. He completed his Engineering and Master's Degree from the Hanoi University of Technology, Vietnam, in 2001 and 2003, respectively. Since 2002 he has been researching in parallel computing and grid computing.

1 Introduction

Service Level Agreements (SLAs), as stated by Sahai et al. (2003), are currently one of the major research topics in grid computing, as they serve as a foundation for reliable and predictable job execution at remote grid sites. SLAs are defined as an explicit statement of expectations and obligations in a business relationship between service provider and customer. Thus, a SLA manages the expectations, regulates resource usage and specifies the costs. The application of such a SLA should guarantee the desired and a-priori negotiated Quality of Service (QoS), which is a mandatory prerequisite for the Next Generation grids. Up to now, some works from Burchard et al. (2004) and Keahey and Motawi (2003) aiming at support SLA for single grid job were noticed.

Besides a vast amount of single applications, which include only one sequent or parallel program, there exist many applications, which require the co-process of many programs following a strict processing order. These applications are called scientific workflow. The scientific workflow fits well with the grid infrastructure as the different resources and expertise distributed across the world can satisfy different requirements of sub-jobs in the workflow. Figure 1 depicts a sample scenario of running a scientific workflow on the grid environment. In a computational grid, each site associated with the system supporting SLA for workflow is a HPCC, which usually has a set of computing nodes, mass storage and a number of experts for technical support. The resource in each HPCC is managed by a software called local RMS. In this paper, the acronym RMS is used to represent the HPCC. As sub-jobs in the workflow must be distributed to different sites, self-managing the grid resources and ensuring the integrity of the workflow during execution is a complex task. For this reason, it is necessary to have a middle component, called workflow broker, handling the task. To provide SLA features, which concentrate on running the workflow on time, the broker must map the sub-jobs of the workflow in such a way that the final workflow deadline will be reached, regardless of the actual execution resource for the sub-jobs and the time needed for the transmission of output/input data between the sub-jobs. To fulfil this requirement, we assume that the user provides the maximal runtime of each sub-job and the underlying RMS supports advance reservations, for example CCS, as described by Hovestadt (2003). Figure 2 depicts a sample CPU reservation profile in such RMS. Queuing-based RMSs are not suitable for our requirements, as no information about the starting time is provided. In our system, we reserve three main types of resources: CPUs, storages and experts.

Figure 1 Sample running workflow scenario

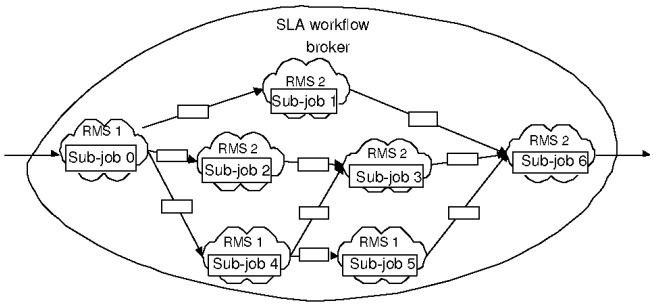
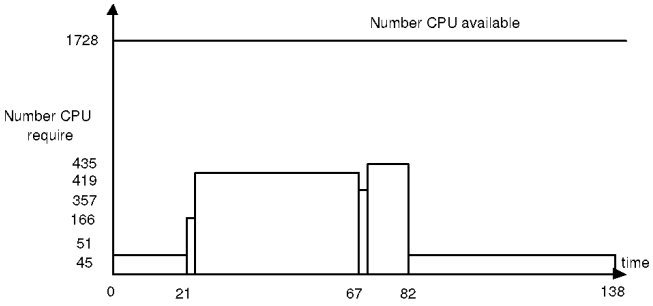


Figure 2 A sample CPU reservation profile of a local RMS

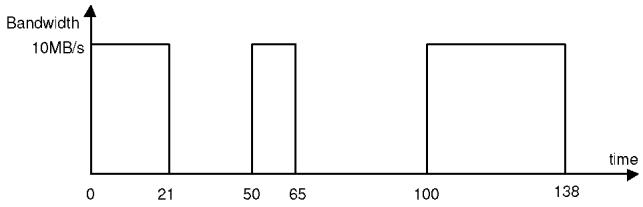


If two sequent sub-jobs are executed in the same RMS, it is not necessary to perform data transferring and the time used for this work is zero. Otherwise, data transferring between them must be performed. As this is a common task in the grid, the bandwidth of the link between two local RMSs also plays an important role in contributing to ensure SLA for the workflow. To make sure that a specific amount of data will be transferred within a specific period of time, the bandwidth must also be reserved. Unfortunately, up to now, there is no mechanism responsible for that task in the worldwide network. Here, to overcome that elimination, we use a central broker mechanism. The link bandwidth between two local RMSs is determined as the average bandwidth between two sites in the network, which has different values for different RMS couples. Whenever having a data transfer task on a link, the SLA broker will determine which time slot is available for the task. During that specified period, the task can use the whole bandwidth and other tasks must wait. Using this principle, the bandwidth reservation profile of a link will look similar to the one depicted in Figure 3. A more correctly model with bandwidth estimation from the works of Wolski (2003) and Wolski et al. (1999) can be used to determine the bandwidth within a specific time period instead of the average value. In both cases, the main mechanism is unchanged.

In a large and complex system like the grid, errors can happen at any time and at any part of the system, with high frequency. The source of errors varies with network cable break, scratched software, hardware failure, etc. This fact carries the potential risk of breaking the negotiated SLA to a system supporting SLA for grid-based workflow and the system must have some preparation for it in order to avoid or eliminate the effect. However, handling all potential errors requires a large amount of effort from the research community over a long time period. In this paper, we concentrate on the case of catastrophic failure, when one or several RMSs are detached from the grid system at a time. When one RMS is detached from the grid system, all running/waiting sub-jobs from several workflows in that RMS are considered failures, as the system cannot control the state and collect the result from them. A checkpoint image of all sub-jobs in the failed RMS cannot be used to restart them in other healthy RMSs. Besides, output data from finished sub-jobs in the fail RMS is also not available. Thus, several waiting sub-jobs in healthy RMSs cannot be run because of unavailable input data. In case of cancelling the workflow because of error, the system will be fined as stated in the SLA. Thus, the system has no way but to try and finish executing the workflow by re-running all failed sub-jobs. However, this task faces two main problems.

- Mapping and re-executing only failed sub-jobs in other healthy RMSs are not enough. Workflow requires a strict execution order to ensure the integrity. Considering only the failure sub-jobs and dismissing others will lead to the potential of breaking integrity character. Thus, determining all sub-jobs which need to continue the workflow execution is a mandatory requirement.
- When sub-jobs in the workflow must be re-executed, the ability to finish the workflow execution on time as stated in the original SLA is very low and the ability to be fined because of not fulfilling SLA is nearly 100%. Within the SLA context, which relates to business, the fine is usually very high and increases with the latency of the workflow's finished time. Thus, those sub-jobs must be mapped to the healthy RMSs in a way, which minimises the workflow finishing time.

Figure 3 A sample bandwidth reservation profile of a link between two local RMSs



This paper presents an error recovery mechanism for grid-based workflows within the SLA context satisfying all the above constraints. It is the next primary progress in a series of efforts, Quan and Kao (2005a, 2005b, 2005c, 2005d), to build a full system supporting SLAs for grid workflows. The paper is organised as follows. Section 2 describes the related work, Section 3 presents the error recovery and Sections 4 and 5 describe the experiment and the implementation architecture, respectively. Section 6 concludes the paper with a short summary.

2 Related work

The importance of fault tolerance in grid computing has already been recognised with the establishment of the grid checkpoint Recovery Working Group. Its purpose is to define user-level mechanisms and grid services for fault tolerance. Stone (2004) described some initial results of the group's effort.

Up to now, other works exploiting the error recovery problem for workflow especially in the SLA context as described in this section have not come to our notice. There is, however, a considerable amount of work in related areas, especially in finding recovery methods for single grid jobs. Garbacki et al. (2005) present a transparent fault tolerance for grid application based on Java RMI. They use globally consistent checkpoints to avoid having to restart long-running computations from scratch after a system crash.

Hwang and Kesselman (2003) present a flexible handling failure framework for the grid. Central to the framework is flexibility in handling failure, which is achieved by using the workflow structure as a high-level recovery policy specification.

Heine et al. (2005a) describe a SLA-aware job Migration in grid environment. The checkpoint of the running job is migrated to the same cluster or another cluster running HPC4U software, which is described by Heine et al. (2005b). An architecture called Virtual Resource Management (VRM) manages and responds when the process takes place smoothly.

Mapping algorithms for grid workflow, which are an important part of the error recovery procedure, receive much attention from the scientific community. In the literature, there are many methods of mapping a grid workflow to grid resources within different contexts. Among them, the old but well-known algorithm Condor-DAGMan from the work of Condor (2004) and Litzkow et al. (1988) is still used in some present grid systems. The algorithm makes local decisions about which job to send to which resource and consider only those jobs which are ready to run at any given instance. This method is not suitable in the context of resource reservation.

Deelman et al. (2003) presented an algorithm, which maps grid workflows onto grid resources based on existing planning technology. This work focuses on coding the problem to be compatible with the input format of specific planning systems and thus transferring the mapping problem to a planning problem. Although this is a flexible way to gain different destinations, which include some SLA criteria, significant disadvantages regarding the time-intensive computation, long response times and the missing consideration of grid-specific constraints appeared.

In recent researches, Ma and Buyya (2005) proposed the x-DCP algorithm, Cooper et al. (2005) and Casanova et al. (2000) proposed the Minmin, Maxmin, suffer algorithms, Blythe et al. (2005) proposed the GRASP algorithm. All of those algorithms concentrate on scheduling the workflow with parameter sweep tasks on grid resources. The common destination of these algorithms is optimising the makespan, defined as the time from when execution starts until the last job in the workflow is completed. Sub-tasks in this kind of workflow can be grouped in layers and there is no dependency among sub-tasks in the same layer. All the proposed algorithms assume each task as a sequence program and each resource as a compute node. By using several heuristics, all these algorithms do the mapping very fast but the quality of the solution is not good enough. Our workflow with DAG form can also be transformed to the workflow with parameter sweep tasks type. A detailed description of the performance of these approaches, as applied to our problem, can be found in Part 4.

With the same resource reservation and workflow model, Quan and Kao (2005d) proposed an algorithm, which maps a workflow to grid resources. The proposed algorithm uses Tabu search to find the best possible assignment of sub-jobs to resources. In order to shorten the computation time caused by the high number of resource profiles to be analysed and by the flexibility while determining start and end times for the sub-jobs, several techniques for reducing the search space are introduced. However, these techniques cannot be applied to solve the problem in this paper. Firstly, the bandwidth reservation model is not considered in the work of Quan and Kao (2005d). Secondly, the destination in our previous work is optimising the cost, while the mapping algorithm in this paper aims at optimising the makespan. However, there is no conflict in the purpose of our work. The previous algorithm is used to map workflow to grid resource in the SLA negotiation phase while the work in this paper is used in the error recovery phase.

3 Error recovery mechanism

In the SLA context, every sub-job of the workflow is planned to run on reserved resources within a specific time period to ensure the QoS while still preserving the integrity of the workflow. An example of such a scenario applied to the example workflow in Figure 1 is presented in Tables 1 and 2. In this paper, time is computed as a time slot. In Table 2, we noticed that with two dependent sub-jobs running in the same RMS, data transfer time between them equal zero.

During the running process of the workflow, one or several RMSs can be detached from the system at any time. We propose a mechanism as described in Figure 4 to detect and recover the error when the problem occurs. Step 1 and the procedure to realise the mechanism, are described in Part 5. In this part, we concentrate on describing the kernel of the error recovery mechanism, which includes Steps 2–4.

Table 1 Sample sub-jobs running time table

ID	*RMS*	*Duration*
0	RMS1	0–5
1	RMS2	7–14
2	RMS2	17–21
3	RMS2	23–30
4	RMS1	7–15
5	RMS1	17–30
6	RMS2	32–35

Table 2 Sample data transfer time table

Sj. link	*RMS link*	*Duration*
0–1	1–2	6–6
0–2	1–2	13–16
0–4	1–1	0
1–6	2–2	0
2–3	2–2	0
4–3	1–2	20–22
4–5	1–1	0
3–6	2–2	0
5–6	1–2	31–31

Figure 4 Abstract error recovery mechanism

1. If error is detected
2. Determine all affected workflows with their associated sub-jobs which need to be remapped.
3. Form new workflows and determine remapping priority for the new workflows.
4. Use w-Tabu algorithm to map each workflow to the healthy RMS.

3.1 Determine sub-jobs which need to be re-planned

Determining all affected sub-jobs in the workflow is done with three following observations.

- Sub-jobs, which are running in the failed RMS, are directly affected and remapping those sub-jobs will lead to the need for remapping all other sequence sub-jobs to ensure the integrity. When the remapping happens, there is no warranty that it will ensure the execution order for other waiting sub-jobs. Beside that, re-planning all waiting sub-jobs in healthy RMSs does not affect the final destination of minimising the workflow execution makespan. Thus, we can say that all directly affected sub-jobs and all waiting sub-jobs in the affected workflow belong to the set of affected sub-jobs and need be re-planned.
- With determined affected sub-jobs in the fail RMSs, they will not have input data to run if their directly finished previous sub-jobs are also in the failed RMSs. Thus, it is necessary to rerun those finished sub-jobs.
- With determined affected sub-jobs in the healthy RMSs, they will not have input data to run if their directly finished previous sub-jobs are in the fail RMSs and the related data transfer task is not finished. Those finished sub-jobs must also be rerun.

From these observations we define the algorithm as presented in Figure 5.

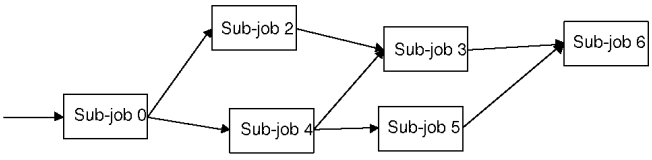
Figure 5 Determining affected sub-jobs algorithm

Determine set of affected workflows, which includes workflow having sub-jobs running/waiting or finished but output data still not transferred in the fail RMSs.
With each one in the set of affected workflow{
Determine the first set of affected sub-jobs, which includes sub-jobs having end_time greater than fail slot
Determine the second set of affected sub-jobs, which includes sub-jobs finished running in the fail RMSs and output data become the input for waiting/ running sub-jobs in the fail RMSs
Determine the third set of affected sub-jobs, which includes sub-jobs finished running in the fail RMSs and output data still not transferred to waiting sub-jobs in the healthy RMSs
The set of affected sub-jobs is formed by combining three above sets }

For illustration purposes, we use the workflow sample in Figure 1 with runtime parameters shown in Tables 1 and 2. Supposing the RMS 1 fails at time slot 10, sub-job 4 is directly affected leading to sub-jobs 3–6 also being affected. When sub-job 3 is re-planned, there is no warranty that it will be run after the finished slot of the waiting sub-job 2. Thus, we determine that all sub-jobs 2–6 in the affected workflow belong to the set of affected sub-jobs and need to be re-planned. Because of the failure of RMS1, output from sub-job 0 is lost and there is no data input for rerunning sub-job 4. Beside that the data transfer task from sub-job 0 to sub-job 2 is also not finished. It is necessary to add sub-job 0 to the list of affected sub-jobs.

In our example, after this phase, all affected sub-jobs of the affected workflow form a new workflow as depicted in Figure 6. These sub-jobs inherit all characters about resource and runtime requirement of the original sub-jobs.

Figure 6 All determined affected sub-jobs in the sample workflow



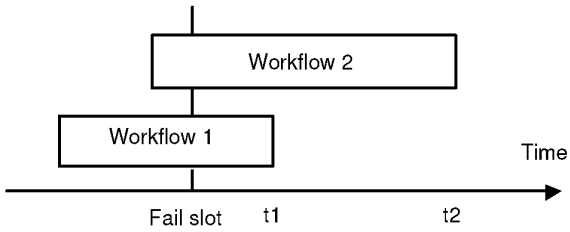
3.2 Determine remapping priority

When the error happens, many workflows can be affected simultaneously and we have to re-plan many new workflows formed from sets of determined affected sub-jobs. One problem appears which, in the newly formed

workflows, should be re-planned earlier. It is important because the workflow, which is mapped earlier, will have lower latency. Here, we use the policy Earliest Deadline First (EDF), which is used broadly in real time systems. Workflows having earlier deadlines will be given higher priority as they occupy shorter resources and the other workflows need a shorter time to wait for available resources. Thus, the total latency is reduced and the fine amount is also reduced.

To clarify the problem, we look at an example as presented in Figure 7. Suppose we have two workflows which need to be re-mapped because of the error and the grid system can execute one workflow at a time. Suppose that the penalty for each late hour is *P*. If the Workflow 2 is mapped first, Workflow 1 has to wait until Workflow 2 is finished. Thus, the minimal fine will be *P*×(t2-failslot). If Workflow 1 is mapped first, the minimal fine will be P×(t1-failslot). Thus, mapping Workflow 1 first is better than mapping Workflow 2. In a real complex situation, mapping Workflow 1 first gives a better chance of finishing Workflow 1 earlier, to release resources earlier and give more chance for Workflow 2 to be mapped with smaller latency.

Figure 7 Scenario of two workflows need to be re-mapped



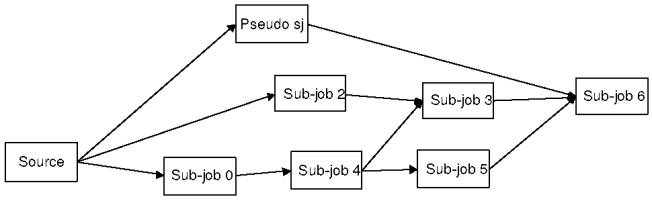
Based on this priority, each workflow will be mapped in sequence with the healthy RMSs. To do the mapping, we refine the new workflow under the Directed Acyclic Graph (DAG) format and then use the mapping module to map this new DAG workflow to RMSs. When forming DAG for a workflow, it is necessary to consider the dependency of affected sub-jobs with running sub-jobs in healthy RMSs to ensure the integrity of the workflow. To present that dependency, in the new workflow, with each running sub-job in the healthy RMSs, we create a pseudo corresponding sub-job with:

- runtime = deadline – fail slot – time overhead
- number of required CPU = 0
- number of required storage = 0
- number of required expert = 0.

where time overhead is the period to do the recovery process. Moreover, we also need a new pseudo source sub-job for the workflow with runtime and resource requirement equal to zero.

In our example scenario, sub-job 6 must run after the finished of sub-job 1 but sub-job 1 does not appear in the set of affected sub-jobs. Thus, we add a pseudo sub-job with runtime = 4 and resource requirement = 0 to the workflow. After this step the newly formed example workflow is as shown in Figure 8.

Figure 8 DAG form of the newly formed workflow



3.3 w-Tabu algorithm

Here, the task is mapping the workflow to the remaining healthy RMSs with the purpose of minimising the makespan. Within the SLA context as defined in part 1, the makespan of the workflow depends on the reservation state of resources in RMSs, bandwidth among RMSs and bandwidth reservation state. It is easy to show that this task is a NP hard problem. Moreover, the defined context in this paper is different from all other contexts appearing in the literature. Thus, the problem needs a new approach in order to be solved. In this paper, we propose a mapping strategy as depicted in Figure 9.

Figure 9 *w*-Tabu algorithm overview

1. Determine assign sequence for all sub-jobs of the workflow
2. Generate reference solution set
3. With each solution in reference set
Use specific procedure based Tabu search to improve the solution as far as possible
4. Pick the solution with best result

This strategy looks similar to an abstract of a long-term local search such as Tabu search, Grasp, SA, etc. However, in order to reach the final destination, several points need to be clarified, which make our distinguish algorithm. The following sessions will describe all the steps in depth.

3.3.1 Determine the assign sequence of the workflow

The assigning sequence of the workflow becomes important when the RMS to execute each sub-job, the bandwidth among sub-jobs, is determined and the next task is determining time slot to run sub-job in the specified RMS. The sequence of determining runtime for sub-jobs of the workflow in RMS can also affect the final makespan, especially in the case of many sub-jobs in the same RMS.

In general, to ensure the integrity of the workflow, sub-jobs in the workflow should be assigned based on the sequence of data processing. However, that principal does not cover the case of a set of sub-jobs, which have the same

priority in data sequence and do not depend on each other. To examine the problem, we determine the earliest and the latest start time of each sub-jobs of the workflow in ideal conditions. The time period to do data transferring among sub-jobs is computed by dividing the amount of data over a fixed bandwidth. The latest start/stop time for each sub-job and data transferring task depends only on the workflow topology and the runtime and not on the resources context. Those parameters can be determined by using conventional graph algorithms. A sample of the data for the newly formed workflow is presented in Table 3.

Table 3 Valid start time for sub-jobs of workflow in Figure 8

Sub-job	*Earliest start*	*Latest start*
Source	0	0
pseudo	0	28
0	0	0
2	11	17
3	20	24
4	8	8
5	19	19
6	34	34

The ability of finding a suitable resource slot to run a sub-job depends on the number of resources that are free during the valid running period. From the graph, we can see sub-jobs 4 and 2 having the same priority in data sequence. However, from the data in Table 3, sub-job 4 can start at max time slot 8 while sub-job 2 can start at max time slot 17 without affecting the finished time of workflow. Suppose that two sub-jobs are mapped to run in the same RMS and the RMS can run one sub-job at a time. If sub-job 2 is assigned first and in the worst case at time slot 17, sub-job 4 will be run from time slot 22. Thus, the workflow will be late at a minimum of 14 time slots. If sub-job 4 is assigned first at time slot 8, sub-job 2 can be run at time slot 18. Thus, the workflow will be late by 1 time slot. Here, we can see that the latest time factor is the main parameter to evaluate the full affection of the sequence assignment decision. It can be seen from the effect, mapping sub-jobs having smaller latest start time first will make the latency smaller. Thus, the latest start time value determined as above can be used to determine the assign sequence. Sub-jobs having smaller latest start time will be assigned earlier.

3.3.2 Generate reference solution set

A solution is found by determining which sub-job of the workflow is run by which RMS. We do not consider the time factor in this phase so a reference solution is defined as a set of map sub-jobs: RMS with all sub-jobs in the workflow. 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-jobs: RMS between two solutions, must be as small as possible. The number of members in the reference set depends on the number of available RMSs and number of sub-jobs. Usually this number is not larger than 30. The algorithm is defined in Figure 10.

Figure 10 Generating reference set algorithm

```
Each candidate RMS of a sub-job has an
assign_number
Each building solution has a similar set, which
contain at least a similar map sub-job : RMS
While size of reference set is not big enough {
  Empty similar set
  With each sub-job in the workflow {
    Find a RMS in the candidate set which
    create minimal number of similar with
    solutions in similar set
    Increase assign_number of the recently
    assigned RMS
    If that assign_number > 1 then find all
    similar defined solution to put to similar set.
}}
```

While building a solution, with each sub-job in the workflow, we select the RMS in the set of candidate RMSs which create the minimal number of similar sub-jobs: RMS with other solutions in the similar set. After that, we search in the set of defined solutions, which have not been in similar sets but satisfy the similar condition with the building solution, and add them to similar sets. When finished, the solution is put to the reference set. After all reference solutions are defined, we use a specific procedure to refine each of the solutions as far as possible.

3.3.3 Solution improvement algorithm

To improve the quality of solution, we use a specific procedure based on short term Tabu search for this problem. But, before anything, we have to determine the time slot for each sub-job as well as the makespan of the present solution. This task is done with the algorithm in Figure 11.

Figure 11 Determining timetable algorithm for workflow

```
With each sub-job k following the assign sequence {
   Determine set of assigned sub-jobs Q, which
   having output data transfer to the sub-job k
   With each sub-job i in Q {
      min_st_tran=end_time of sub-job i +1
      Search in reservation profile of link between
      RMS running sub-job k and RMS running sub-
      job i to determine start and end time of data
      transfer task with the start time > min_st_tran
   }
   min_st_sj=max end time of all above data
   transfer +1
   Search in reservation profile of RMS running
   sub-job k to determine its start and end time with
   the start time > min_st_sj
}
```

For each sub-job of the workflow in the assigning sequence, we first find all the runtime periods of data transfer tasks from previous sub-jobs to current sub-jobs. 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 later than the latest finish time of previous related data transfer task.

In normal Tabu search, in each move iteration, we will try assigning each sub-job $sj_{ik} \in Gsj_i$ with each RMS r_i in the candidate set R_{ik} and use the procedure in Figure 11 to compute the runtime and then check for overall improvement and pick the best one. This method is not effective because it requires a lot of time for computing the runtime of the workflow, which is not simple. We will improve the method by proposing a new neighbourhood with two comments.

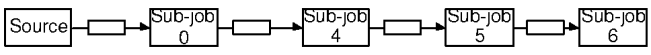
Comment 1. The runtime of the workflow depends mainly on the execution time of the Critical Path (CP). In one iteration, we can move only one sub-job to one RMS. If the sub-job does not belong to the CP, after the move, the old CP will have a very low probability of being shortened and the makespan of the workflow has low probability of improvement. Thus, we concentrate only on sub-jobs in the CP. With a defined solution and timetable, the CP of a workflow is defined with the algorithm described in Figure 12.

Figure 12 Determining Critical Path (CP) algorithm

```
Let C is the set of sub-jobs in the critical path
Put last sub-job into C
next_sub-job=last sub-job
do{
   prev_sub-job is determined as the sub-job having
   latest finished data output transfer to
   next_sub-job
   Put prev_sub-job into C
   next_sj=prev_sub-job
} until prev_sj= first sub-job
```

We start with the last sub-job determined. The next sub-job of the CP will have latest finish data transfer to the previous determined sub-job. The process continues until the next sub-job = first sub-job. Figure 13 depicts a sample CP of the workflow in Figure 7.

Figure 13 A sample Critical Path (CP)



Comment 2. In one move iteration, with only one change of one sub-job to one RMS, if the finish time of the data transfer from this sub-job to the next sub-job in the CP is not decreased, the CP can not be shortened. For that reason, we only consider the change, which shortens the finish time of the consequent data transfer. It is easy to see that checking if the data transfer time is much shorter than computing the timetable for the whole workflow can improve.

With two comments and other remaining procedure similar to the standard Tabu search, we built the overall improvement procedure as presented in Figure 14.

Figure 14 The solution improvement algorithm

```
Start from specific solution and time table
While not reach max_loop{
   determine critical path
   with each sub-job in the critical path {
     with each RMS in the RMS candidate set {
       if can improve the finished time of the
       sequence data transfer {
         compute time table with new solution
         store tuple (sub-job, RMS, workflow finish
         time) to candidate list
   } } }
   Pick the best solution satisfy aspiration condition
   or not affect tabu rule
   Aspiration condition is workflow finish earlier.
   Assign tabu_number for the selected RMS
   If have improvement then store the solution
}
```

4 Performance experiment

The performance experiment is done with simulation to check for the quality of the *w*-Tabu algorithm in Part 3.3 and the overall performance of the mechanism. The hardware and software used in the experiments is rather standard and simple (Pentium 42.8 Ghz, 1 GB RAM, Linux Redhat 9.0, MySQL). The total simulation program is about 10,000 lines of C/C++ code.

4.1 Mapping algorithm performance experiment

The goal of the experiment is to measure the feasibility, the quality of the solution and the time needed for the computation. Up to now, we have not noticed a similar resource model or workflow model as stated in Part 1, thus we do not have previous works to compare the result with. To overcome this problem, we employ all the ideas from a recently published paper by Ma and Buyya (2005), Cooper et al. (2005), Casanova et al. (2000) and Blythe et al. (2005) related to mapping workflow to grid resource with the same destination, to minimise makespan and adapt them to our problem. Those algorithms include *w*-DCP, Grasp, Minmin, Maxmin and suffer. In the following we will describe those adaptation algorithms for our problem.

4.1.1 Minmin algorithm

Cooper et al. (2005) and Casanova et al. (2000) described the Minmin algorithm. It uses the Minimum Minimum Completion Time (MCT) as a metric, meaning that the task that can be completed the earliest is given priority.

The motivation behind Minmin is that assigning tasks to hosts that will execute them the fastest will lead to an overall reduced makespan. The adaptation of the Minmin algorithm is presented in Figure 15.

Figure 15 Minmin algorithm

```
Analyse the workflow into set of sub-jobs in
sequence layers. Sub-jobs in the same layer not
depend on each other.
With each layer in sequence {
   while set of sub-jobs in the layer not empty {
     with each sub-job in the layer {
       Determine the RMS which can finish the sub-
       job in earliest end_time
       Store tuple (sub-job, RMS, end\_time) in a list
     }
   Pick (sub-job, RMS, end_time) with min
   end_time
   Drop the selected sub-job out of layer
 }}
```

4.1.2 Maxmin algorithm

Max-min's metric is the Maximum MCT. The expectation is to overlap long-running tasks with short-running ones as stated by Cooper et al. (2005); Casanova et al. (2000). The algorithm is described in Figure 16.

Figure 16 Maxmin algorithm

```
Analyse the workflow into set of sub-jobs in
sequence layers. Sub-jobs in the same layer not
depend on each other.
With each layer in sequence {
   while set of sub-jobs in the layer not empty {
     with each sub-job in the layer {
       Determine the RMS which can finish the sub-
       job in earliest end_time
       Store tuple (sub-job, RMS, end\_time) in a list
     }
   Pick (sub-job, RMS, end_time) with max
   end_time
   Drop the selected sub-job out of layer
 }}
```

4.1.3 Suffer 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. The algorithm is depicted in Figure 17.

4.1.4 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. We apply all the algorithms without changing anything so it is not necessary to represented that algorithm here. More details can be seen in the work of Blythe et al. (2005).

Figure 17 Suffer algorithm

```
Analyse the workflow into set of sub-jobs in
sequence layers. Sub-jobs in the same layer not
depend on each other.
With each layer in sequence {
   while set of sub-jobs in the layer not empty {
     with each sub-job in the layer {
       Determine the RMS which can finish the sub-
       job in earliest end_time
       Determine the second earliest end_time
       suffrage value=the earliest end_time - the
       second earliest end_time
       Store tuple (sub-job, RMS, suffrage value) in
       a list
     }
     Pick (sub-job, RMS, suffrage value) with min
     suffrage value
     Drop the selected sub-job out of layer
}}
```

4.1.5 w-DCP algorithm

The DCP algorithm is based on the principle of continuously shortening the longest path (also called CP) in the task graph, by scheduling tasks in the current CP to an earlier start time. This principal was applied for scheduling workflows with parameter sweep tasks on global grids by Ma and Buyya (2005). Directly applying this algorithm to our problem faces many difficulties, especially in determining parameters Absolute Earliest Finished Time (AEFT), Absolute Latest Finished Time (ALFT). For this reason we propose a new algorithm called *w*-DCP as presented in Figure 18.

Figure 18 w-DCP algorithm

```
Initialise initial solution, compute time table,
determine the critical path.
Loop until the makespan can not be decreased {
  While all sub-jobs in the critical path is not
  marked {
    Determine the critical path
    with each unmarked sub-job in the critical path{
      with each RMS candidate of that sub-job {
        if first sub-job compute end_time of the
           sequence data transfer task
        else compute the end_time of the sub-job
        push tuple (sub-job, RMS, end_time) to list
      }
      Pick the result with min end_time
      Marked the sub-job
}}}
```

To compare the quality of all described algorithms above, we generated 90 different workflows which:

- Having different topologies.
- Having different number of sub-jobs. Number of sub-jobs is in the range 7–32.
- Having different sub-job specifications.
- Having different amount of data transfer.

We stop at 32 sub-jobs for a workflow because as far as we know, with our model of parallel task sub-job, most existing scientific workflows as described by Ludtke et al. (1999), Berriman et al. (2003) and Richter (2004) just include 10–20 sub-jobs. With each workflow, we do mapping to 20 RMSs with different resource configurations in nine different start times, that means, with nine different resource scenarios. The total of running instances for each algorithm is 810. The quality of the solution is determined by the makespan of the workflow. The experimental results show that our algorithm needs a maximum of 13 seconds to map a workflow to the resource. Presenting all experiment data in this paper is impossible because of the space limitation, so we describe only some in the entire data. Figure 19 and Table 4 present a sample detailed experiment data mapping workflow with 21 sub-jobs to 20 RMSs. Each row in Table 4 presents the start time slot and the end time slot of each solution found by each algorithm, respectively. Table 5 views some of the experimental results in average relative value of the solution makespan from other algorithms as compared to our algorithm. Figure 20 depicts the overall comparison among algorithms.

Figure 19 Work flow with 21 sub-jobs

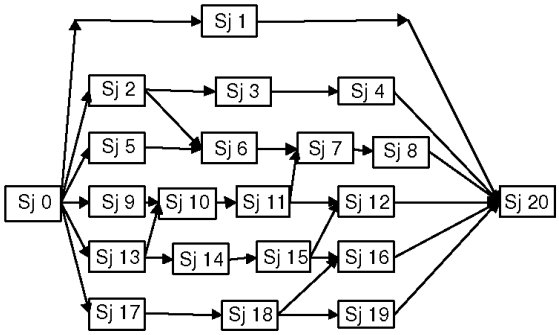


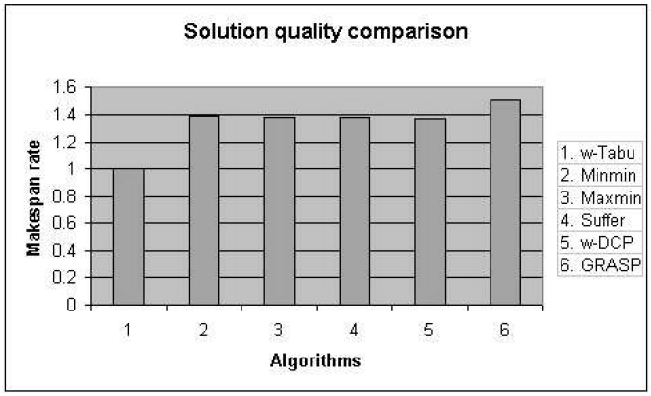
Table 4 Experimental results with workflow containing 21 sub-jobs

St.	*w-TB*	*w-DCP*	*GRASP*	*Minmin*	*Maxmin*	*Suffer*
10	127	132	145	132	132	132
15	132	136	150	139	137	137
20	137	137	155	144	142	143
25	142	146	162	149	146	142
30	147	155	161	151	151	151
35	152	152	170	156	156	156
40	157	159	172	161	161	161
45	162	169	173	172	172	162
50	167	175	181	177	177	167

Table 5 Some experimental results in relative value

Id	*w-TB*	*w-DCP*	*GRASP*	*Minmin*	*Maxmin*	*Suffer*
1	1.00	1.49	1.48	1.48	1.47	1.55
6	1.00	1.47	1.46	1.45	1.46	1.56
10	1.00	1.51	1.50	1.50	1.49	1.59
15	1.00	1.47	1.48	1.47	1.47	1.61
20	1.00	1.44	1.45	1.46	1.47	1.57
25	1.00	1.42	1.41	1.42	1.41	1.51
30	1.00	1.45	1.43	1.43	1.42	1.51
35	1.00	1.45	1.44	1.44	1.46	1.60
40	1.00	1.40	1.38	1.39	1.38	1.50
45	1.00	1.40	1.38	1.39	1.39	1.52
50	1.00	1.33	1.32	1.33	1.33	1.44
55	1.00	1.32	1.31	1.31	1.31	1.44
60	1.00	1.34	1.33	1.32	1.32	1.45
65	1.00	1.33	1.32	1.31	1.34	1.50
70	1.00	1.30	1.31	1.30	1.30	1.47
75	1.00	1.30	1.30	1.31	1.32	1.47
80	1.00	1.30	1.29	1.29	1.30	1.44
85	1.00	1.26	1.25	1.26	1.24	1.37
90	1.00	1.21	1.20	1.19	1.19	1.34

Figure 20 Overall quality comparison



From the experiment data, it can be seen that our algorithm outperforms all other algorithms in every case. When runtime is just a few seconds, the algorithm is effective enough to be used in real system.

4.2 Error recovery performance experiment

The goal of the experiment is to measure the total reaction time of the recovery mechanism in absolute value when the error happens. Determining the total reaction time is important because it helps to define the earliest start time of the re-map workflow, which is a necessary parameter for mapping algorithms. To do the experiment, we use 20 RMSs with different resource configurations and then we fill all the RMSs with randomly generated workflows having start time slots equal to 20. The number of failing RMSs increases from 1 to 3, which RMS fails is selected randomly. With each number of failing RMS, fail slot is increased along the reservation axis. The reason for this activity is that the error can happen at

any random time slot along the reservation axis; thus, the broader range of experimental time is, the more correctly reaction time value can be determined. At each time, we used the described recovery mechanism to re-map all affected workflow as well as all affected sub-jobs and measured runtime. The runtime is computed in seconds.

Table 6 presents the experimental data when one RMS fails. As can be seen from the data, the total reaction time of the mechanism increases following the increase in the total number of affected sub-jobs. When the number of failure RMSs increases the total number of affected sub-jobs increases but the number of healthy RMSs decreases. For this reason, the total reaction time of the mechanism when the number of failing RMSs is increasing does not have a big difference compared to the case of having 1 RMS failure as presented in Tables 7 and 8. Further, the probability of having more than two RMS failures simultaneously is very small. For these reasons, the simulation data can be depended upon. With the total reaction time just less than 2 minutes compared to hourly running workflow, the performance of the algorithm is well accepted in real situations. In the mapping algorithm, time is computed in slots, which can have a resolution from 2 minutes to 4 minutes. The reaction time of the mechanism will occupy one time slot and the time for the system to do negotiations takes about one time slot; thus the start time slot of the re-map workflow can be assigned to the value of present time slot plus two.

Table 6 Experimental results with one failure RMS

Fail-slot	*Total-wf.*	*Total-sj.*	*Runtime*
50	10	156	90
60	10	143	74
70	10	138	66
80	10	135	55
90	10	117	44
100	10	109	45
110	10	98	36
120	10	89	29

Table 7 Experimental results with two failure RMSs

Fail-slot	*Total-wf*	*Total-sj*	*Runtime*
50	13	199	80
60	13	180	63
70	13	170	56
80	13	152	48
90	13	140	47
100	13	129	46
110	13	115	35
120	13	103	29

Table 8 Experimental results with three failure RMSs

Fail-slot	*Total-wf.*	*Total-sj.*	*Runtime*
50	14	213	79
60	14	194	71
70	14	183	61
80	14	164	54
90	14	152	51
100	14	140	46
110	14	127	41
120	14	115	34

5 System integration

An implemented prototype provides basic features, which allow any client to negotiate SLAs, monitor running workflows and receive the result. Based on standard components such as Globus Toolkit 3.2, MySQL database, Maui ME for the local RMS, the system is compatible with the existing grid infrastructure. An error recovery module was integrated into the overall system architecture as depicted in Figure 21. However, this module does not work alone or just plug loosely to the system; it interacts with other modules to create a concrete system and can handle the defined error. The co-operation scheme among many system modules to perform error recovery procedures is presented in Figure 22. The following parts will describe in detail the working process of cooperation in the system. Detailed description about other modules in the overall system architecture is presented in Quan and Kao (2005a, 2005c).

Figure 21 Implemented system architecture

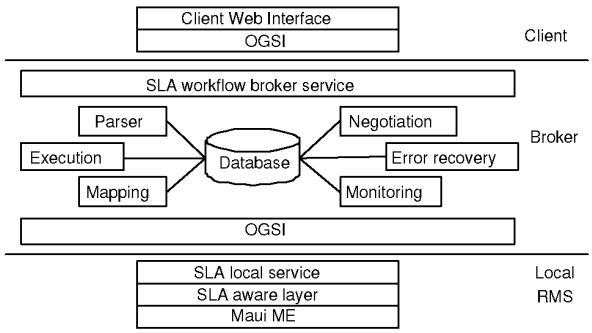
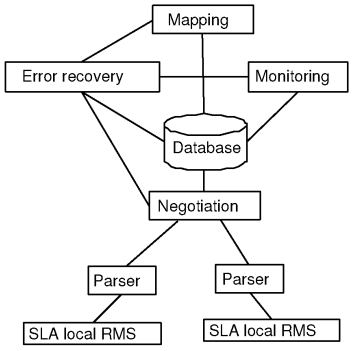


Figure 22 Interaction among modules to perform error recovery



5.1 Error detection

Error detection is done with a monitoring module. A monitoring module collects information about RMS state, RMS resource, RMS reservation, sub-jobs state, etc., from all RMS. The information is analysed and stored in the central database to ensure that the broker module could have an overall picture of the system. With the pull model, after a period of time, the broker will connect with local RMSs and request monitoring information. If a broker cannot contact a local RMS it will consider that RMS as failed and activate the error recovery module.

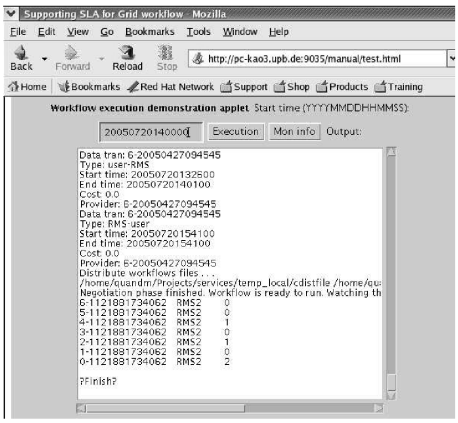
5.2 Error recovery procedure

When error recovery module is activated, it will perform following actions in strict sequence:

- Access database to retrieve information about failure RMSs and determine affected workflow as well as necessary sub-jobs of the workflow to be remapped.
- Based on determined information about affected workflows and sub-jobs, activate negotiation module to cancel all SLA sub-jobs with local RMSs related to specific sub-jobs. All negotiation activities are done with the help of SLA text as the means of communication.
- Activate monitoring module to update newest information about RMS, especially information about resource reservation.
- Call mapping modules to determine where and when sub-jobs in the affected workflow will be run.
- Based on mapping information, activate the negotiation module to sign new SLAs for each sub-job with the specific local RSM.
- Update workflow control information, sub-jobs information in the central database.

The online demonstration of the execution of a workflow consisting of seven sub-jobs in cooperation of three local RMS can be found at http://pc-kao3.upb.de:9035/manual/test.html. Figure 23 depicts the web-based user interface in the running process.

Figure 23 Initial web based client



6 Conclusions

This paper described an error recovery method for the workflow within the SLA context. We do not have the ambition of covering all cases of errors, which can happen to the system supporting SLAs for workflow but concentrate on the catastrophic scenario where one or several RMSs are detached from the system at a time. The main contribution of the paper is located in the newly stated problem and the proposed mechanism to solve it. We proposed an algorithm to detect all affected sub-jobs when the error happens and an algorithm to re-map those sub-jobs to the remaining healthy RMSs with makespan optimisation. We also adapted many recently formulated and related algorithms to our problem in order to prove the performance as well as the quality of our strategy. All the results were integrated into our existing prototype system, which can be demonstrated online.

Future work will concentrate to building mechanisms to react with other cases of errors in order to have a reliable and stable grid system.

Acknowledgements

The work reported in this paper has been partially supported by PC2, with Odej Kao as the Director. The authors thank Odej Kao for many helpful discussions.

References

Berriman, G.B., Good, J.C., Laity, A.C., Bergou, A., Jacob, J., Katz, D.S., Deelman, E., Kesselman, C., Singh, G., Su, M-H. and Williams, R. (2003) 'Montage: a grid enabled image mosaic service for the national virtual observatory', *ADASS*, Vol. 13, pp.135–138.

Blythe, J., Jain, S., Deelman, E., Gil, Y., Vahi, K., Mandal, A. and Kennedy, K. (2005) 'Task scheduling strategies for workflow-based applications in grids', *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)*, 759–767.

Burchard, L., Hovestadt, M., Kao, O., Keller, A. and Lin-nert, B. (2004) 'The virtual resource manager: an architecture for SLA-aware resource management', *Proceeding of the IEEE CCGrid 2004*, IEEE Press, Chicago, Illinois, USA, pp.126–133.

Casanova, H., Legrand, A., Zagorodnov, D. and Berman, F. (2000) 'Heuristics for scheduling parameter sweep applications in grid environments', *Proceeding of the 9th Heterogeneous Computing Workshop (HCW, 2000)*, IEEE Press, Cancun, Mexico, pp.292–300.

Condor (2004) Version 6.4.7 Manual www.cs.wisc.edu/condor/manual/v6.4, 10 December.

Cooper, K., Dasgupta, A., Kennedy, K., Koelbel, C., Mandal, A., Marin, G., Mazina, M., Mellor-Crummey, J., Berman, F., Casanova, H., Chien, A., Dail, H., Liu, X., Olugbile, A., Sievert, O., Xia, H., Johnsson, L., Liu, B., Patel, M., Reed, D., Deng, W., Mendes, C., Shi, Z., YarKhan, A. and Dongarra, J. (2005) 'New grid scheduling and rescheduling methods in the GrADS project', *International Journal of Parallel Programming*, Vol. 33, pp.209–229.

Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Blackburn, K., Lazzarini, A., Arbree, A., Cavanaugh, R. and Koranda, S. (2003) 'Mapping abstract complex workflows onto grid environments', *Journal of Grid Computing*, Vol. 1, pp.25–39.

Garbacki, P., Biskupski, B. and Bal, H. (2005) 'Transparent fault tolerance for grid application', *Proceedings of the European Grid Conference (EGC 2005),* LNCS, Amsterdam, Netherland, Vol. 3470, pp.671–680.

Heine, F., Hovestadt, M., Kao, O. and Keller, A. (2005a) 'Provision of fault tolerance with grid-enabled and SLA-aware resource management systems', *Proceeding of the Parallel Computing Conference Parco 2005*, John von Newman Press, Malaga, Spain, pp.113–120.

Heine, F., Hovestadt, M., Kao, O. and Keller, A. (2005b) 'SLA-aware job migration in grid environments', in Grandinetti (Ed.): *Grid Computing: New Frontiers of High Performance Computing*, Elsevier Press, The Netherlands, pp.185–201.

Hovestadt, M. (2003) 'Scheduling in HPC resource management systems: queuing vs. planning', *Proceeding of the 9th Workshop on JSSPP at GGF8*, LNCS, Seattle, WA, USA, pp.1–20.

Hwang, S. and Kesselman, C. (2003) 'Grid workflow: a flexible failure handling framework for the grid', *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, IEEE Press, Seattle, WA, USA, pp.126–131.

Keahey, M. and Motawi, K. (2003) 'The taming of the grid: virtual application service', *Argonne National Laboratory Technical Memorandum*, Vol. 262, pp.134–143.

Litzkow, M., Livny, M. and Mutka, M. (1988) 'Condor a hunter of idle workstations', *Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS 88)*, IEEE Press, San Jose, California, pp.104–111.

Ludtke, S., Baldwin, P. and Chiu, W. (1999) 'EMAN: semi-automated software for high-resolution single-particle reconstruction', *Journal of Structure Biology*, Vol. 128, pp.82–97.

Ma, T. and Buyya, R. (2005) '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, Rio de Janeiro, Brazil, pp.251–258.

Quan, D.M. and Kao, O. (2005a) 'On architecture for an SLA-aware job flows in grid environments', *Journal of Interconnection Networks, World Scientific Computing*, Singapore, pp.245–264.

Quan, D.M. and Kao, O. (2005b) 'SLA negotiation protocol for Grid-based workflows', *Proceedings of the International Conference on High Performance Computing and Communications (HPPC-05)*, LNCS, Sorento, Italia, Vol. 3726, pp.505–510.

Quan, D.M. and Kao, O. (2005c) 'On architecture for an SLA-aware job flows in grid environments', *Proceedings of the 19th IEEE International Conference on Advanced Information Networking and Applications (AINA 2005)*, IEEE Press, Taipei, Taiwan, pp.287–292.

Quan, D.M. and Kao, O. (2005d) 'Mapping grid job flows to grid resources within SLA context', *Proceedings of the European Grid Conference (EGC 2005),* LNCS, Amsterdam, Netherland, Vol. 3470, pp.1107–1116.

Richter, L. (2004) 'Workflow support for complex grid applications: integrated and portal solutions', *Proceeding of the European Across Grids Conference*, LNCS, Nicosia, Cyprus, Vol. 3165, pp.129–138.

Sahai, A., Graupner, S., Machiraju, V. and Moorsel, A. (2003) 'Specifying and monitoring guarantees in commercial grids through SLA', *Proceeding of the 3rd IEEE/ACM CCGrid2003*, IEEE Press, Tokyo, Japan, pp.292–300.

Stone, N. (2004) *Gwd-I: An Architecture for Grid Checkpoint Recovery Services and a Gridcpr Api.*, http://gridcpr.psc.edu/GGF/docs/draft-ggf-gridcpr-Architecture-2.0.pdf, 10 December.

Wolski, R. (2003) 'Experiences with predicting resource performance on-line in computational grid settings', *ACM SIGMETRICS Performance Evaluation Review*, Vol. 30, No. 4, pp.41–49.

Wolski, R., Spring, N. and Hayes, J. (1999) 'The network weather service: a distributed resource performance forecasting service for metacomputing', *Journal of Future Generation Computing System*, Vol. 15, Nos. 5–6, pp.757–768.