Mapping of SLA-based workflows with light communication onto Grid resources

Dang Minh Quan[1] and Jörn Altmann[1 2]

[1] International University of Bruchsal, Campus 3, 76646 Bruchsal, Germany
dang.minh@i-u.de
[2] TEMEP, School of Engineering, Seoul National University, South-Korea
jorn.altmann@acm.org

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

1 Introduction

Service Level Agreements (SLAs) define a business contract between users and service providers. SLAs in the context of Grid computing describe an business environment, in which providers guarantee users that their workflow will be executed on the Grid within the agreed upon period of time and will get paid for the service usage. One of the main system components, which support SLAs management for Grid-based workflows is the job mapping mechanism. The mapping mechanism that we will present maps each sub-job of the workflow to resources in a manner that satisfies two main criteria: First, it will guarantee the workflow execution on time; Second, it will calculate a low cost solution.

If a customer uses a service, he is charged based on the service usage and can expect a quality as indicated in the SLA. An automated mapping is necessary as it frees users from the tedious job of assigning sub-jobs to resources under many constraints such as workflow integrity and execution deadline, cost minimisation. Additionally, a good mapping mechanism will help users to minimize the cost for using Grid resources.

We have developed a running system for SLA-based workflows [7–10], which differs from other works [5, 6] in the following aspects. In our system, sub-jobs of the workflow can be sequential or parallel programs and each resource can manage many sub-jobs at the same time. Our work on this topic solved the problem for workflows with heavy communication. The key difference of this paper and the previous works is the communication. The light communication can help us

ignore the complex in time and cost caused by data transfer. Thus, we could apply specific technique to improve the speed and the quality of the mapping algorithm. An initial solution for the case of workflows with light communication has been presented in [7], but without considering variable runtime of sub-jobs and different resource requirements.

This paper, which belongs to this series of efforts to develop a job mapping framework for SLA-based workflows [7–10], will present a solution for variable runtime of sub-jobs and different resource requirements.

Like many popular systems handling SLA-based workflows [1–3], our workflow system is of the Directed Acyclic Graph (DAG) form. It is assumed that the data to be transferred among sub-jobs is very small, usually less than 10MB. The user is also required to specify the estimated runtime of each sub-job together with the specific resource requirements. The time is split into slots. Each slot represents a constant period of real time, in the range of 2 to 5 minutes. Figure 1 illustrates an example of a Grid workflow with resource requirements as specified in Table 1. The label of each link of the workflow represents the amount of data, which has to be transferred between two sub-jobs.

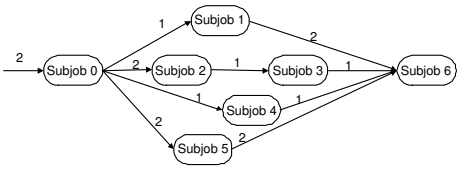

Fig. 1. A sample workflow

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

Table 1. Resource requirements for sub-jobs

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

HPCCs are usually inter-connected by 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 transferred through a link within one time slot can range from 1.2GB to 3GB. Since we assume less than 10MB of data transfer

between sub-jobs (workflows with light communications), the data transfer can easily be performed within one time slot (right after the sub-job had finished its calculation) without affecting any other communication between two RMSs.

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

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

Based on the given input, we are looking for a feasible and, possibly, cost-minimizing solution. The required solution is a set defined in Formula 1.

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

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:

- **Criteria1:** All $K_i \neq \emptyset$. There is at least one RMS in the candidate set of each sub-job.
- **Criteria2:** The total runtime period of the workflow must be within the expected period given by the user.
- **Criteria3:** The dependency among the sub-jobs is resolved and the execution order remains unchanged.
- **Criteria4:** Each RMS provides a profile of currently available resources and can run many sub-jobs of a single workflow in parallel. Those sub-jobs, which run on the same RMS, form a profile of resource requirements. For each RMS r_j with its profile of available resources and for each time slot, , the number of available resources must be larger than the resource requirements.
- **Criteria5:** The data transmission task e_{ki} from sub-job s_k to sub-job s_i takes one timeslot right after the finished time of sub-job s_k , $e_{ki} \in E$. If sub-job s_k and sub-job s_i are executed in the same RMS the data transfer task is neglected. This can be assumed since all compute nodes in a cluster usually use a shared storage system like NFS or DFS. Thus, the time to move data in the same storage is zero.

In the next phase, feasible solutions with the lowest cost are sought. As the number of data to be transferred between sub-jobs in the workflow is very small,

we can omit the cost of data transfer. Thus, the cost C of a Grid workflow is defined in Formula 3. It is the sum of the charge of using: (1) the CPU, (2) the storage and (3) the expert knowledge.

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

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

It can easily be shown that the cost-minimizing mapping of a workflow onto RMSs is a NP-hard problem.

2 Related works

In [5, 6], Zeng et al and Iwona et al built systems to support QoS features for SLA-based workflows. To map a workflow onto a Grid resource, they used Integer Programming (IP) for finding a solution. Applying IP to our problem is impossible because of two reasons. First, the arbitrary length of a sub-job cannot be expressed in an Integer Programming model. The time to complete a sub-job depends on the resource configuration and the reservation profile of the RMS. Second, an RMS can handle many parallel programs at the same time. Thus, we do not know how many, which, and when sub-jobs will be executed in an RMS. Consequently, we cannot formulate the IP constraint of available resource as described in Criteria 4.

Meta-heuristics such as Genetic Algorithm and Simulated Annealing proved to be very effective in mapping and scheduling problems. However, in the case of our problem, with the appearance of resource profiles, the evaluation at each step of the search becomes expensive for large problems [10].

A mechanism (based on Tabu search) for mapping a light communication workflow onto Grid resources is described in [7]. In order to shorten the computation time caused by the high number of resource profiles to be analyzed and by the wider range of start time of the sub-job without violating the end time, several techniques for reducing the search space were introduced. However, these techniques cannot be applied to solve the problem in this paper because of the variable runtime lengths of the sub-jobs within different RMSs.

The work in [10] solved the problem of mapping a heavy communication workflow onto Grid resources. The H-Map algorithm has been proposed. The main idea of the H-Map algorithm is forming a widely distributed set of initial configurations and performing the local search for each of them. As the problem in [10] is closed to the problem in this paper, we can adapt the H-Map algorithm to map the light communication workflow onto Grid resources. The framework is retained as described in Figure 2, but we change the computing timetable function and computing cost function to suit the new requirement.

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

Fig. 2. Framework of the H-Map algorithm

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

Fig. 3. The application of DBC algorithm to our problem

The original DBC Grid scheduling algorithm [11], called the cost-time optimization scheduling algorithm, is used to schedule parameter sweep application on global Grids. This algorithm builds on the cost-optimization and time-optimization scheduling algorithms. This is accomplished by applying the time-optimization algorithm for scheduling task-farming jobs onto distributed resources having the same processing cost. Even this algorithm only supports sequential sub-jobs. However, the idea can also be applied to our problem since our workflow can be considered a parameter sweep application. The modified algorithm is presented in Figure 3.

3 L-Map algorithm

In this paper, we propose an algorithm called L-Map to map light communication workflows onto the Grid RMSs (L - stands for light). The goal of the L-Map algorithm is to find a solution, which satisfies the requirements as described in Section 1.

Each sub-job has different resource requirements. There are a lot of RMSs with different resource configurations. The initial action is to find the suitable RMSs among those heterogeneous RMSs, which can meet the requirement of the sub-job. The matching between the sub-job's resource requirements and the RMS's resource configuration is done by several logic checking conditions in the WHERE clause of the SQL SELECT command. This work will satisfy Criteria 1. Suppose that each sub-job has m RMSs in the candidate list, we could have m^n configurations. The overall L-Map algorithm is presented in Figure 4. The following sections will describe each procedure of the algorithm in detail.

Step 0: With each sub-job s_i, we sort the RMSs in the candidate set K_i according to the cost of running s_i. The cost is computed according to Formula 3. The configuration space of the sample is presented in Table 2. Each RMS-Rt column presents the RMS and the runtime of the sub-job in this RMS.

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

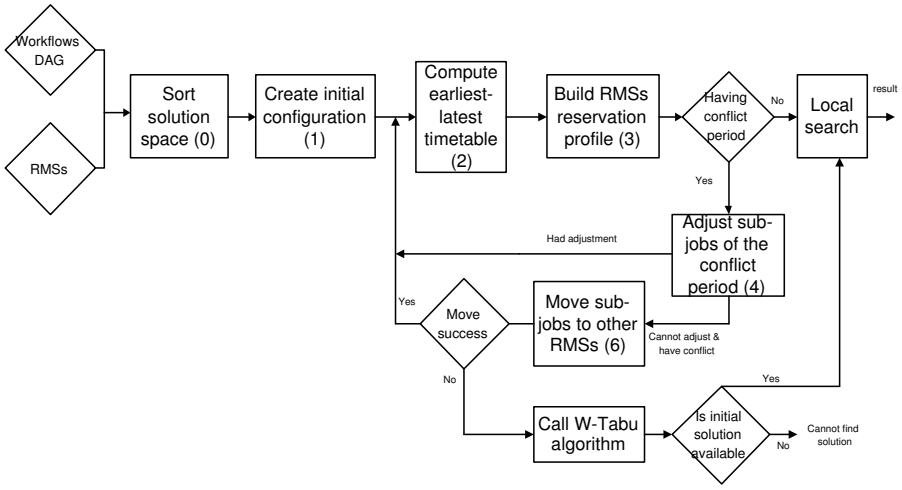

Fig. 4. Framework of the L-Map algorithm

Sj_ID	RMS-Rt	RMS-Rt	RMS-Rt
0	R1 - 16	R3 - 17	R2 - 16
1	R1 - 14	R2 -14	R3 - 14
2	R1 - 16	R2 - 16	R3 - 17
3	R1 - 16	R3 - 15	R2 - 17
4	R1 - 15	R2 - 17	R3 - 15
5	R1 - 14	R3 - 16	R2 - 16
6	R1 - 16	R3 - 16	R2 - 17

Table 2. RMSs candidate for each sub-job in cost order

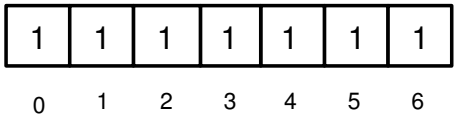

Fig. 5. The first selection configuration of the example

is described as a vector. The index of the vector represents the sub-job and the value of the element represents the RMS. The first configuration in our example is illustrated in Figure 5.

Step 2: As the runtime of each sub-job in the selected RMS was defined and the time to do data transfer equals zero, we can compute the earliest start time and the latest stop time of each sub-job using conventional graph algorithms. Following this procedure, we ensure that the Criteria 2 is met. In the case of our example, we assume that the user wants the workflow to be started at time slot 10 and stopped at time slot 85. The resulting Earliest-Latest timetable is shown in Table 3.

Step 3: For each RMS appearing in a configuration and each type of resource in the RMS, we build the resource reservation profile using the Earliest-Latest timetable. In this step, the runtime of the sub-job is computed from the earliest start time to the latest stop time. In our example, only RMS1 appears in the configuration. The CPU reservation profile of the RMS 1 is presented in Figure

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

Table 3. The earliest-latest timetable

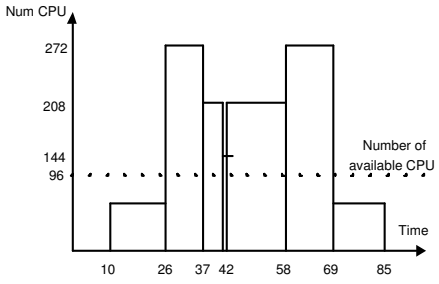

Fig. 6. Reservation profile of RMS 1

6. As can be seen from Figure 6, there are many conflict periods, in which, the number of required resources is greater than the available resources.

Step 4: We move sub-jobs out of the conflict period by adjusting the earliest start time or the latest stop time of the sub-jobs. One possible solution for our example is shown in Figure 7(a), where either the latest stop time of subjob1 is set to t1 or the earliest start time of subjob2 is set to t2. The second possible solution is to adjust both sub-jobs simultaneously as depicted in Figure 7(b). A necessary prerequisite here is that after adjusting, the latest stop time minus earliest start time of the sub-job is larger than its runtime.

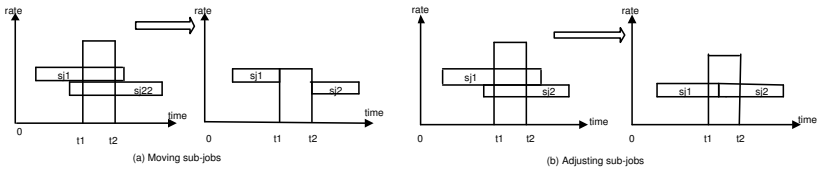

Fig. 7. Resolving the conflict period

Step 5: We adjust the earliest start time and latest stop time of the sub-jobs that are linked with the moved sub-jobs and then repeat step 3 and 4 until we cannot adjust any sub-jobs any more. The results of this step applied to our example, the CPU reservation profile of RMS 1, is depicted in Figure 8.

Step 6: If there are still some conflict periods after step 5, we have to move some sub-jobs contributing to the conflict to other RMSs. However, the resource, which has the conflict period, should be utilized as much as possible. Thus, the cost for using the resources will be kept at a minimum. This is a knapsack problem, which is known to be NP-hard. Therefore, we use the heuristic algorithm of Figure 9. This algorithm ensures that the remaining free resources are always less than the smallest sub-job. If a sub-job cannot be moved to another RMS, we can deduce that the Grid resource is busy and the w-Tabu algorithm [9] is invoked. w-Tabu algorithm maps a SLA-based workflow to Grid resources with

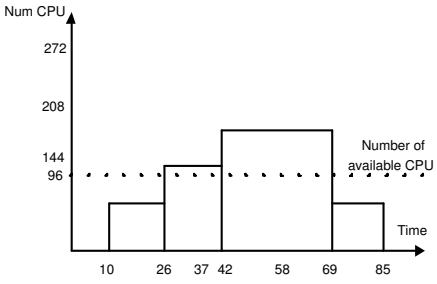

Fig. 8. Reservation profile of RMS 1 after adjusting

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

Fig. 9. Moving sub-jobs algorithm

makespan optimization. If the w-Tabu cannot find an solution, the algorithm will stop.

In the case of our example, the largest conflict period is 42-69 with allocations of sub-job 3, 5 and 4 (sorted in descending order according to the cost). Since we can fill the period with sub-job 3 only, sub-job 4 is moved to RMS 2 and sub-job 5 is moved to RMS 3. This step created a new configuration.

Step 7: As we created a new configuration, the process from step 3 to step 6 is repeated until there is no conflict period. This process will satisfy the conditions of Criteria 3, 4 and 5. After this phase, we have a feasible candidate solution.

Step 8: A local search procedure is used to improve the quality of the solution as far as possible. We search in the neighborhood of the candidate solution for a better solution. If a better solution has been found then it will play the role of the candidate. The process is repeated until we cannot find a better solution. More details about how this procedure meets all the criteria can be seen in [10].

4 Performance experiment

The goal of the experiment is to measure the feasibility of the solution, its cost, and the time needed for the computation. The hardware used for the experiments is rather standard and simple (Intel Duo 2,8Ghz, 1GB RAM, Linux FC5). To conduct the experiments, 18 workflows with different topologies, number of sub-jobs, sub-job specifications, and amount of data transferred between sub-jobs, are generated. These workflows are then mapped to 20 RMSs with different resource configurations and resource reservation profiles by 3 algorithms: L-Map, H-Map, and DBC. In the experiment, 30% number of RMS having CPU performance equals to the requirement, 60% number of RMS having CPU performance is 100% more powerful than requirement, 10% number of RMS having CPU performance is 200% more powerful than requirement. Along with the increasing in performance, the price for each CPU class is also increased. The runtime of each sub-job in each type of RMS is assigned by using formula 4.

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

with pk_i, pk_j being the performance of a CPU in RMS r_i, r_j respectively and rt_i being the estimated runtime of the sub-job with the resource configuration of RMS r_i. k is the speed up control factor. 60% number of sub-jobs having $k = 0.5$, 30% number of sub-jobs having $k = 0.25$, 10% number of sub-jobs having $k = 0$. The description about resource configurations and workload configurations can be seen at the address: http://it.i-u.de/schools/altmann/DangMinh/desc_expe1.txt. The final result of the experiment is presented in Table 4. Column Sjs (Sub-jobs) presents the number of sub-jobs within the workflows. The cost of the solution and the runtime for finding the solutions for each algorithm are recorded in column Runtime and column Cost.

	L-Map		H-Map		DBC	
Sjs	Cost	Runtime	Cost	Runtime	Cost	Runtime
Simple level experiment						
7	625.386634	1	625.386634	1	625.386634	0.5
8	749.706622	0.5	749.706622	0.5	755.306618	0.5
9	768.416626	0.5	768.416626	1	774.016622	0.5
10	830.883292	0.5	833.683290	1	833.683290	0.5
11	883.923294	0.5	883.923294	1	883.923294	0.5
12	936.393294	0.5	936.393294	2	936.393294	0.5
13	992.233294	0.5	992.233294	1	992.233294	0.5
Intermediate level experiment						
14	1044.093300	1	1044.093300	2	1049.693296	0.5
15	1169.953288	1	1169.953288	2	1175.553284	0.5
16	1294.273276	0.5	1294.273276	2	1294.333288	0.5
17	1353.649942	0.5	1353.649942	3	1353.649942	0.5
18	1477.969930	0.5	1477.969930	2	1477.969930	0.5
19	1501.396596	1	1501.396596	7	1504.196594	0.5
20	1560.773262	0.5	1560.773262	3	1560.773262	0.5
21	1584.153262	1	1584.153262	6	1584.153262	0.5
Advance level experiment						
25	1762.433256	2	1762.433256	5	1770.833250	0.5
28	1862.649928	3	1862.649928	7	1873.849920	0.5
32	2184.839906	4	2190.499914	10	2193.239900	0.5

Table 4. Performance experiment result

The experiments are divided into 3 levels. Within the simple level, 7 workflows with a number of 7 to 13 sub-jobs are mapped onto 20 RMSs. The result shows that the solutions created by different algorithms are identical for workflows with the same number of sub-jobs.

In the intermediate level experiment, we map 8 workflows that have 14 to 19 sub-jobs. The result of this experimentshow a difference in the quality of the solution found by the different algorithms. Method DBC, which do not use local search and need relative smaller runtime, found lower quality solution than other methods. Method H-Map found high quality solutions but needed more time than L-Map.

The advance level experiment mapped 3 workflows (with the number of sub-jobs in the range from 25 to 32). The result of this experiment shows that L-Map algorithm found out higher quality solutions than the DBC algorithm. It also found equal or even better solutions in a shorter time than the H-Map algorithm.

5 Conclusion

This paper has presented an algorithm, which performs a cost-efficient and fast allocation of workflows with light communication between sub-jobs onto Grid resources with respect to SLA-defined runtime constraints. In our work, the distinguishing factor is that the number of data to be transferred between sub-jobs is very small. Considering this factor allows to detect and resolve the conflict periods quickly. The simulation-based performance evaluation showed that the proposed algorithm creates solutions of equal or better quality than existing algorithms. Besides, it needed a significantly smaller computation time than the other two algorithms in our comparative study. These characteristics are positive factors for applying the method in real environments.

References

1. E. Deelman, J. Blythe, Y.Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, S. Koranda: Mapping Abstract Complex Workflows onto Grid Environments. Journal of Grid Computing, Vol 1, no. 1, (2003) 25-39.
2. J. Cao, S. A. Jarvis, S. Saini, G. R. Nudd: GridFlow: Workflow Management for Grid Computing. Proc. 3rd IEEE/ACM Int. Symp. on Cluster Computing and the Grid, Tokyo, Japan, (2003) 198-205.
3. R. Lovas, G. Dzsa, P. Kacsuk, N. Podhorszki, D. Drtos: Workflow Support for Complex Grid Applications: Integrated and Portal Solutions, Proc. 2nd European Across Grids Conference, Nicosia, Cyprus, (2004).
4. A. Sahai, V. Machiraju, M. Sayal, L. J. Jin, F. Casati: Automated sla monitoring for web services. DSOM 2002, LNCS 2506, (2002) 28–41.
5. L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, H. Chang: QoS-Aware Middleware for Web Services Composition. IEEE Transactions on Software Engineering, v.30 n.5, (2004) 311–327.
6. I. Brandic and S. Benkner and G. Engelbrecht and R. Schmidt: QoS Support for Time-Critical Grid Workflow Applications, Proc. e-Science (2005).
7. D.M. Quan, O. Kao: Mapping Grid job flows to Grid resources within SLA context. Proc. the European Grid Conference,(EGC 2005), LNCS 3470, (2005) 1107–1116.

8. D.M. Quan, O. Kao: On Architecture for an SLA-aware Job Flows in Grid Environments. Journal of Interconnection Networks, World scientific computing, (2005) 245–264.
9. D.M. Quan: Error recovery mechanism for Grid-based workflow within SLA context. To be published by International Journal of High Performance Computing and Networking (IJHPCN), (2007).
10. D.M. Quan, D.F. Hsu: Mapping Heavy Communication Grid-Based Workflows onto Grid Resources Within An SLA Context Using Metaheuristics. To be published by International Journal of High Performance Computing and Application (IJHPCA), (2007).
11. 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), Vol 35, no 5, (2005) 491–512.
12. M. Hovestadt, O. Kao, A. Keller, A. Streit: Scheduling in HPC Resource Management Systems: Queuing vs. Planning. Proc. 9th Workshop on JSSPP at GGF8, LNCS 2862, (2003) 1-20.
13. D.P. Spooner, S.A. Jarvis, J. Cao, S. Saini, G.R. Nudd: Local grid scheduling techniques using performance prediction. IEEE Proc. Computers and Digital Techniques, (2003) 87–96.