Mapping a group of jobs in the error recovery of the Grid-based workflow within SLA context

Dang Minh Quan
International University in Germany
School of Information Technology
Bruchsal 76646, Germany
quandm@upb.de

Jorn Altmann
International University in Germany
School of Information Technology
Bruchsal 76646, Germany
jorn.altmann@acm.org

Abstract

The error recovery mechanism receives an important position in the system supporting Service Level Agreements (SLAs) for the Grid-based workflow. If one sub-job of the workflow is late, a group of directly affected sub-jobs should be re-mapped in a way that does not affect the start time of other sub-jobs in the workflow and is as inexpensive as possible. With the distinguished workload and resource characteristics as well as the goal of the problem, this problem needs new method to be solved. This paper presents a mapping algorithm, which can cope with the problem. Performance measurements deliver good evaluation results on the quality and efficiency of the method.

1. Introduction

Service Level Agreements (SLAs) [13] are currently one of the major research topics in Grid Computing, as they serve as a foundation for a reliable and predictable job execution at remote Grid sites. We have developed the system supporting SLA for the Grid-based workflow [8, 10, 9]. The scenario of running a Grid-based workflow is presented in Figure 1.

Like many popular systems handling Grid-based workflows [3, 14, 6], our proposed workflow system is of the Directed Acyclic Graph (DAG) form. It is noted that a sub-job of the workflow can be either a sequential program or a parallel program and that the data to be transferred among sub-jobs can be enormous.

In each Grid site, the resources are managed by the software called local Resource Management System (RMS). Each RMS has its own unique resource configuration. To ensure that the sub-job can be executed within a dedicated time period, the RMS must support advance resource reservation such as CCS [5].

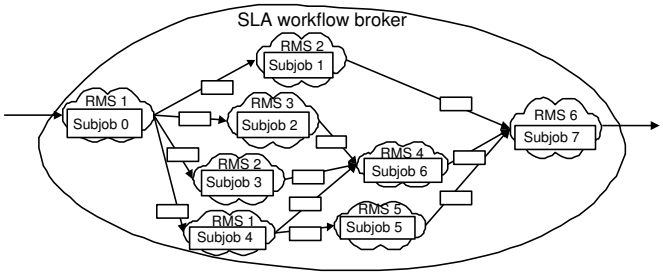

Figure 1. A sample running Grid-based workflow scenario

If two sequential sub-jobs are executed in the same RMS, it is not necessary to do data transfer and the time used for this work equals zero. Otherwise, data transfer tasks between the RMS's must be performed. To make sure that a specific amount of data will be transferred within a specific period of time, a bandwidth must also be reserved. The mechanism to reserve the bandwidth is described in [11, 12].

During the sub-job running process, an error inside an RMS may happen at any time and can damage the whole workflow. The error can be node scratch, hardware failure, network cable break. In this case, the RMS will restart the sub-job from the checkpointing image. The time overhead to detect the error and the time to rerun the sub-job from the checkpointing image will make the finished time of the sub-job later than the pre-determined deadline. When one sub-job is late, its output data transfer to the next sub-jobs cannot be done. Therefore, those next sub-jobs cannot run in the right way because of lacking input data and the whole wokflow is damaged. For example, if sub-job 0 is late 1 time slot, sub-job 1, 2, 3 and 4 are also affected because of having no input data.

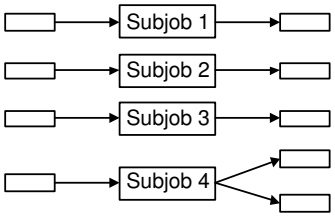

Figure 2. Affected sub-jobs and data transfers

Therefore, we have to try to re-map the directly affected sub-jobs in a way that does not affect the start time of other remaining sub-jobs in the workflow. When we re-map the directly affected sub-jobs, we also have to re-map their related data transfers. With the example in Figure 1, if sub-job 0 is late, the affected sub-jobs and data transfers are described in Figure 2. This task can be feasible because of many reasons.

- The latency period is very small, only 1 or 2 time slots.
- The Grid may have others solutions that the data transfers will be shorter because the links have broader bandwidth.
- The Grid may have RMSs with higher CPU power, which can execute the sub-jobs in shorter time.

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 the given workflow.
- Let Sa be the set of all directly affected sub-jobs with the resource and runtime requirements.
- Let E be the set of data transfer in the given workflow.
- Let Ei be the set of input data transfers of S.
- Let Eo be the set of input data transfers of S.
- 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, $K_i \subset R$, $s_i \in Sa$.

Based on the given input, a feasible and possibly optimal solution is sought, which allows the most efficient mapping of those sub-jobs in a Grid environment with respect to the given deadlines. The required solution is a set defined as Formula 1.

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

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

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

A feasible solution must satisfy following conditions:

- **Criteria1:** All $K_i \neq \emptyset$. There is at least one RMS in the candidate set of each sub-job.
- **Criteria2:** The start time of each input data transfer ei_h must later than the sub-job it depends on, $ei_h \in Ei$. The stop time of each output data transfer eo_k must earlier than the next sub-job which depends on it, $eo_k \in Eo$.
- **Criteria3:** Each RMS provides a profile of currently available resources and can run many sub-jobs both sequentially and parallel. Those sub-jobs, which run on the same RMS, form a profile of resource requirement. With each RMS r_j running sub-jobs of Sa, 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.
- **Criteria4:** The data transmission task e_{ki} from sub-job s_k to sub-job s_i must not overlap other reserved data transmission task on the link between RMS running sub-job s_k to RMS running sub-job s_i, $e_{ki} \in Ei \cup Eo$, $s_k, s_i \in Sa$.

In the next phase the feasible solution with the lowest cost is sought. The cost C of a Grid workflow is defined in formula 3, 4 and 5. It is a sum of four factors: the cost of (1) using the CPU, (2) storage, (3) expert knowledge, and (4) data transfer between resources.

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

$$C_2 = \sum e_{ki}.n_d * r_j.p_d \quad (4)$$

$$C = C_1 + C_2 \quad (5)$$

with $s_i.r_t, s_i.n_c, s_i.n_s, s_i.n_e$ are the runtime, number CPU, number storage, number expert of sub-job s_i respectively. $r_j.p_c, r_j.p_s, r_j.p_e, r_j.p_d$ are the price 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.

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 shown easily that the optimal mapping of those sub-jobs to Grid RMSs with cost optimizing is a NP hard problem.

This paper presents a mapping algorithm called G-map to handle this problem. It is the next progress in a series of efforts, [8, 10, 9, 7, 11, 12], to build a full system supporting SLAs for Grid-based workflows.

2. Related work

Our problem can be defined as a special case of the mapping a bag of independent tasks to resources problem [15, 4, 1]. In the literature, most of the efforts [15, 4, 1] solve this problem with time optimization. It is the major difference to our work because we concentrate to find out a solution which meets the deadline and optimizes the cost.

The most closed work to our problem is the work from [2]. In [2], the authors present the method to schedule parameter sweep applications on global Grids. The original deadline and budget constrained (DBC) costtime optimization algorithm builds on the cost-optimization and time-optimization scheduling algorithms. This is accomplished by applying the time-optimization algorithm to schedule task-farming application jobs on distributed resources having the same processing cost. The authors assume that all tasks are sequential programs and identical to each other. It is clear that our problem is distinguished from parameter sweep applications scheduling problem as the sub-jobs in our problem are parallel programs with various differences in configurations as well as in constraints. However, the primary idea of DBC algorithm can also be applied to our problem as presented in Figure 3.

```
For each sub-job in the set {
   Sort the candidate RMSs according to cost order. If 2 or
    more RMSs have the same cost, the more powerful RMS
    will stay first
    For each RMS in the sorted candidate list {
        calculate the execution time of that sub-job on the
         RMS. If it meet the deadline then assigned the sub-
         job to the RMS
}}
```

Figure 3. Application of DBC algorithm to the problem

In [12], we present an algorithm called H-Map to map heavy communication workflow to the Grid resources. The character of resources and workload are similar to this problem. Therefore, we can easily adapt H-Map to the problem as described in Figure 4. The main idea of H-Map algorithm is that a set of initial configuration distributed over the search space according to cost factor will be further refined to find the best solution.

```
Create set of reference configurations distributed over the
search space according to cost factor
for each sub-job in the set {
    for each RMS in the candidate list {
        if cheaper then put (sjid, RMS id, improve_value)
        to a list }}
sort the list according to improve_value
from the begin of the list{
    Compute time table to get the finished time
    If finished time < limit
        break
}
Store the result
```

Figure 4. Application of H-Map algorithm to our problem

As the number of affected sub-jobs is not always big we can apply the searching all cases algorithm (SAC). In the case of having small number of affected sub-jobs, the runtime of the algorithm is sufferable. The solution result of this algorithm can be a good reference source to evaluate the quality of other algorithms. The searching all cases algorithm is presented in Figure 5.

```
min_cost=1000000
With each possible configuration a {
    determine the scheduling order
    check the feasibility of a
    if feasible {
        compute the cost m_cost of a
        if(m_cost<min_cost){
            min_cost=m_cost
            store a
} } }
```

Figure 5. Searching all cases algorithm

3. G-map algorithm

G-Map algorithm maps a group of sub-jobs on to the Grid resources with G stands for Group. In the G-Map algorithm, we try to compress the solution space in a way that the ability to have feasible solutions is higher. After that, a set of initial configuration is constructed. This set will be improved by local search until it cannot be improved any more. Finally, we pick the best solution from the final set. The architecture of the algorithm is presented in Figure 6.

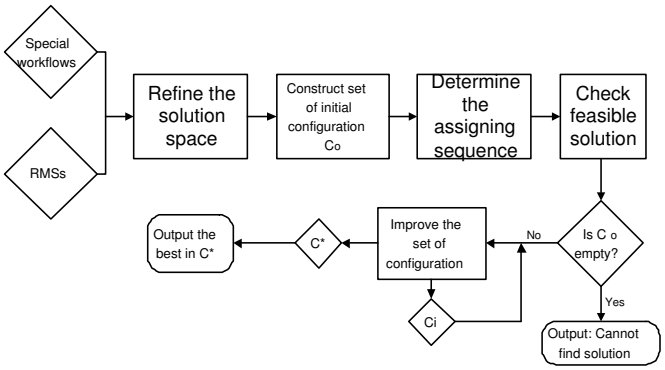

Figure 6. G-Map algorithm

3.1. Refining the solution space

The set of candidate RMSs for each sub-job can be continuously refined by following observation: An RMS will be valid with a sub-job only if the sub-job assigned to that RMS satisfies the start time of the next sequential sub-jobs. The algorithm to refine the solution space is presented in Figure 7.

With each separate sub-job, we determine the schedule time of the input data transfers, the sub-job and output data transfer. From the algorithm in Figure 7, we can see that the resource reservation profile is not updated. We call this the ideal assignment. If the stop time of the output data transfer does not earlier than the start time of the next sequential sub-job then we remove the RMS out of the candidate set.

3.2. Constructing the set of initial configurations

The goal of the algorithm is finding out a feasible solution, which satisfies all required criteria and is as inexpensive as possible. Therefore, the set of initial configurations should satisfy two criteria.

```
for each sub-job k in the set {
  for each RMS r in the candidate list of k{
    for each link to k in assigned sequence{
      min_st_tran=end_time of source sub-job
      search reservation profile of link  the
      start_tran > min_st_tran
      end_tran = start_tran+num_data/bandwidth
      update reservation profile
    }
    min_st_sj=max (end_tran)
    search in reservation profile of  r the
    start_job > min_st_sj
    end_job= start_job + runtime
    for each link from k in assigned sequence{
      min_st_tran=end_job
      search reservation profile of link  the
      start_tran > min_st_tran
      end_tran = start_tran+num_data/bandwidth
      update reservation profile
      if end_tran>=end_time of destination sub-job
        remove r       out of the candidate list
}}}
```

Figure 7. Refining the solution space procedure

- The configurations in the set must differ from each other as far as possible. This criterion will ensure that the set of initial configuration will distribute widely over the search space.
- The RMS running sub-job in each configuration should differ from each other. This criterion will ensure that each sub-job will be assigned in ideal condition, thus the ability to become a feasible solution will increase.

The procedure to create the set of initial configuration is as following.

Step1: Sorting the candidate set according to the cost factor. With each sub-job, we compute the cost of running the sub-job by each RMS in the candidate set and then sort the RMSs according to the cost. With the case of our example, we could have a sorted solution space as presented in Figure 8.

Step2: Forming the first configuration. The procedure to form the first configuration in the set is presented in Figure 9. We form the first solution having as small cost as possible. With each unassigned sub-job, we compute the m_delta = cost running in the first feasible RMS minus cost running in the second feasible RMS in the sorted candidate list. The sub-job having the smallest m_delta will be assigned to the first feasible RMS. The purpose of this action is to en-

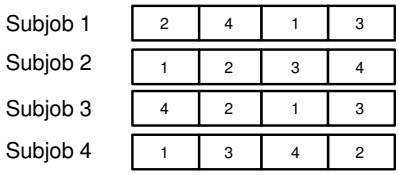

Figure 8. A sample sorted solution space

sure that the sub-job having higher ability to increase the cost will be assigned first. After that, we will update the reservation profile and check if the assigned RMS is still available for other sub-jobs. If not we will mark it as unavailable. The process is repeated until all sub-jobs are assigned. The selection of which sub-job to be assigned is effective when there are many sub-jobs having the same RMS as the first feasible solution.

```
While the set of unassigned sub-jobs is not empty {
   Foreach sub-job s in the set of unassigned sub-jobs {
      m_delta=cost in first feasible RMS- cost in second
      feasible RMS
      put (s, RMS, m_delta) in a list
   }
   Sort the list to get the minimum m_delta
   Assign s to the RMS
   Drop s out of the set of unassigned sub-jobs
   Update the reservation profile of the RMS
   Check if the RMS is still feasible with other unassigned
   sub-jobs
   if not, mark the RMS is infeasible
}
```

Figure 9. The algorithm to form the first configuration

Step3: Forming the other configurations. The procedure to form other initial configurations is described in Figure 10. To satisfy two criteria as described above, we use $assign\_number$ to keep track of number of the assignment RMS to a sub-job and l_ass to keep track of the appearance frequency of RMS within a configuration. RMS having smaller $assign\_number$ and small appearance frequency in l_ass will be selected. Applying this algorithm to the example, starting from the first configuration in Step 2, the process of forming other initial configurations is described in Figure 11.

```
assign_number of each candidate RMS =0
While number of configuration < max_sol {
  clear list of assigned RMS l_ass
  for each sub-job in the set {
     find in the candidate list RMS r having the
     smallest number of appearance in l_ass
     and the smallest assign_number
     Put r to l_ass
     assign_number++
}}
```

Figure 10. Procedure to create the initial configuration set

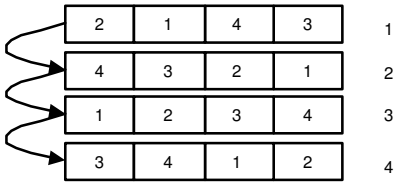

Figure 11. The sample initial configuration set

3.3. Determining the assigning order

When the RMS executing each sub-job and the bandwidths among sub-jobs were determined, the next task is determining time slot to run a sub-job in the specific RMS. At this time, the order of determining scheduled time for sub-jobs becomes important. The sequence of determining runtime for sub-jobs in RMS can also affect the Criteria 2 especially in the case of having many sub-jobs in the same RMS. In this algorithm, we use the policy like in [12]. Thus, the input data transfer having the earliest start time smaller will be scheduled earlier. The output data transfer having the latest stop time smaller will be scheduled earlier. The sub-job having earlier deadline should be scheduled earlier.

3.4. Checking the feasibility of a solution

To check the feasibility of a solution we have to determine the timetable to execute sub-jobs and their related data transfer. In the error recovery phase, finding a solution that meet the Criteria 2 is very important. Therefore, we do not simply use the provided runtime of each sub-job but modify it according to the performance of each RMS. Let pk_i, pk_j is the performance of a CPU in RMS r_i, r_j respectively and $pk_j > pk_i$. Suppose that a sub-job has the provided runtime rt_i with the resource requirement equals to r_i. Thus, the run-

time rt_j of the sub-job in r_j is determined as in Formula 6.

$$rt_j = \frac{rt_i}{\frac{pk_i+(pk_j-pk_i)*k}{pk_i}} \quad (6)$$

Parameter k presents the affection of the sub-job's communication character and the RMS's communication infrastructure. For example, if pk_j equals to $2*pk_i$ and rt_i is 10 hours, rt_j will be 5 hours if k equals to 1. However, k=1 only when there are no communication among parallel tasks of the sub-job. Otherwise, k will be less than 1. Parameter k_a is an average value, which is determined by the user through many experiments and is provided as the input for the algorithm. In the reality environment, k may fluctuate around the average value depending on the network infrastructure of the system. For example, suppose that k_a equals to 0.8. If the cluster has good network communication, the real value of k may increase to 0.9. If the cluster has not so good network communication, the real value of k may decrease to 0.7. Nowadays, with the very good network technology in High Performance Computing Centers, the fluctuation of k is not so much. To overcome the fluctuation problem , we use the pessimistic value k_p instead of k in the Formula 6 to determine the new runtime of the sub-job as following.

- If $k_a > 0.8$, for example with the rare communication sub-job, $k_p = 0.5$.
- If $0.8 > k_a > 0.5$, for example with normal communication sub-job, $k_p = 0.25$.
- If $k_a < 0.5$, for example with heavy communication sub-job, $k_p = 0$.

The pessimistic policy will ensure that the sub-job can be finished within the new determined runtime period. With those assumption, the procedure to determine the timetable is presented in Figure 12.

After determining the timetable, the stop time of the output data transfer will be compared with the start time of the next sequential sub-jobs. If having a violation, this solution is determined infeasible.

3.5. Improving solution quality algorithm

If the initial configuration set $C_0 \neq \emptyset$, the set will gradually be refined to have better quality solutions. The refining process stops when the solutions in the set cannot be improved any more and we have the final set $C*$. The best solution in $C*$ will be output as the result of the algorithm. More detail about this procedure can be found in [12].

```
for each sub-job k in the set {
  for each link to k in assigned sequence{
     min_st_tran=end_time of source sub-job
     search reservation profile of link  the
     start_tran > min_st_tran
     end_tran = start_tran+num_data/bandwidth
     update link reservation profile
  }
  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
  update resource reservation profile
  for each link from k in assigned sequence{
    min_st_tran=end_job
    search reservation profile of link  the
    start_tran > min_st_tran
    end_tran = start_tran+num_data/bandwidth
    update link reservation profile
}}
```

Figure 12. Procedure to determine the timetable

4. Performance experiment

Performance experiment is done with simulation to check for the quality of the G-Map algorithm. The goal of the experiment is comparing the quality of G-Map algorithm with other algorithms in different workloads and resource contexts. The quality of an algorithm is evaluated by several factors: the ability of finding feasible solution, the cost of the found solution, the execution time. The hardware and software used in the experiments are rather standard and simple (Pentium D 2,8Ghz, 1GB RAM, Fedora Core 5, MySQL). The total simulation program includes about 5000 lines of C/C++ code.

To compare the quality of all described algorithms above, we generated 8 different workflows which have different topologies, different sub-job specifications, different amount of data transfers, different the maximum number of the potential directly affected sub-jobs, from 1 to 10.

As the difference in the static factors of an RMS such as OS, CPU speed and so on can be easily filter by SQL query, we use 20 RMSs with the resource configuration equal or even better than the requirement of sub-jobs. Those RMSs have already had some initial workload in their resource reservation profiles and bandwidth reservation profiles. Those 8 workflows are mapped to 20 RMSs. We select the late sub-job in each workflow in a way that the number of the directly af-

fected sub-jobs equals the maximum number of the potential directly affected sub-jobs of that workflow. The late period is 1 time slot. With each group of the affected sub-jobs, we change the power configuration of RMS and the k value of affected sub-jobs. Those configurations are presented in Table 1. For example, with the first row in the Table 1, the resource configuration 90-0-10 means that there is 90% number of RMS having CPU performance like requirement, 0% number of RMS having CPU performance 25% more power than requirement, 10% number of RMS having CPU performance 50% more power than requirement. The workload configuration 90-0-10 means that 90% number of affected sub-jobs having $k = 0.5$, 0% number of affected sub-jobs having $k = 0.25$, 10% number of affected sub-jobs having $k = 0$.

Resource configuration			Workload configuration		
0%	25%	50%	k=0.5	k=0.25	k=0
90	0	10	90	0	10
90	10	0	90	10	0
60	30	10	60	30	10
60	10	30	60	10	30
30	60	10	30	60	10
30	10	60	30	10	60
10	30	60	10	30	60
10	60	30	10	60	30
10	0	90	10	0	90
10	90	0	10	90	0
0	10	90	0	10	90
0	90	10	0	90	10

Table 1. Resource configuration scenario and workload configuration scenario

With each affected sub-job group, with each power resource configuration scenario, with each workload configuration scenario, we do mapping with 4 algorithms: G-Map, DBC, H-Map, search all cases. Thus, with each algorithm, we have total 8*12*12=1152 running instances. With each running instance we record the runtime of the algorithm and the cost of the solution if it is feasible.

Table 2 presents the detail cost and runtime of each algorithm for different group of affected sub-jobs in three extreme experimental scenario. Among 4 algorithms, only the SAC algorithm have great runtime when the size of the problem increase. The runtime of this algorithm becomes exponent when number of affected sub-job greater than or equal 6. Other algorithms have very small runtime. In all cases, the runtime of H-Map, DBC, G-Map is not greater than 1 second.

	SAC		H-Map		DBC		G-Map	
Sjs	Cost	Rt	Cost	Rt	Cost	Rt	Cost	Rt
Resource configuration 90-10-0 and Workload configuration 0-10-90								
3	50.3	4	50.3	0.5	50.3	0.5	50.3	0.5
4	62.0	59	62.0	1	62.0	1	62.0	1
5	87.7	1211	87.7	1	87.7	1	87.7	1
6	-	-	78.4	1	81.5	1	78.5	1
7	-	-	95.1	1	102.3	1	95.1	1
8	-	-	113.5	1	113.5	1	113.5	1
9	-	-	110.7	1	119.7	1	110.7	1
10	-	-	144.6	1	150.6	1	142.8	1
Resource configuration 60-30-0 and Workload configuration 30-60-0								
3	50.3	4	50.3	0.5	50.3	0.5	50.3	0.5
4	N/S	60	N/S	1	N/S	1	N/S	1
5	87.7	1217	87.7	0.5	87.7	1	87.7	1
6	-	-	81.5	1	92	0.5	81.5	1
7	-	-	95.1	1	102.3	1	95.1	0.5
8	-	-	N/S	0.5	113.5	0.5	113.5	1
9	-	-	N/S	1	N/S	1	N/S	0.5
10	-	-	N/S	0.5	166.2	1	153.6	1
Resource configuration 0-10-90 and Workload configuration 90-10-0								
3	50.3	4	50.3	0.5	50.3	0.5	50.3	0.5
4	N/S	57	N/S	1	N/S	1	N/S	1
5	87.7	1205	87.7	0.5	87.7	1	87.7	1
6	-	-	81.5	1	92	0.5	81.5	1
7	-	-	95.1	1	102.3	1	95.1	1
8	-	-	N/S	0.5	N/S	0.5	N/S	1
9	-	-	N/S	1	N/S	1	N/S	1
10	-	-	N/S	0.5	166.2	1	153.6	1

Table 2. Performance result in three extreme experimental scenario

From the data in the table, we can see clearly a trend that if the Grid has a lot of more powerful RMSs than requirement and the group of affected sub-jobs has a lot of computing intensive jobs (big k), the chance to have a feasible solution will be higher and vice versa. Because of having the greatest runtime, the search all cases algorithm can find out solution within an acceptable period when the size of the problem is small. Therefore, it has the smallest ability to find a feasible solution. H-Map algorithm also has limited ability to find a feasible solution. The reason is that H-Map is designed for mapping the whole workflow but not the special case like this problem. Thus, there are a lot of infeasible solutions in the initial configuration set. G-Map and DBC algorithm have the same ability to find

a feasible solution. Figure 13 presents the total number of finding out feasible solution in the whole experiment co-relative with each algorithm.

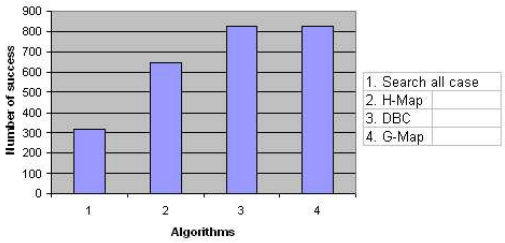

Figure 13. Total number of feasible found solution by algorithms

From the data of the experiment, we can see the domination of local search approach to the quality of the solution. If H-Map or G-Map algorithm can find a feasible solution, it is high quality solution with low cost. As search all case and H-Map have smaller ability to find a feasible solution. We only compare the quality of the solution between G_Map and DBC algorithms. The experimental data shows that G-Map find lower cost solution than DBC. The average cost in relative value between G-Map and DBC is 1 versus 1.05.

5. Conclusion

This paper has presented a method, which performs an efficient and precise assignment of a group of sub-jobs in the error recovery of the Grid-based workflow within SLA context with respect to ensure the start time of other sub-jobs and cost optimization. In our work, the distinguished character is that those sub-jobs of the workflow can be a sequential or parallel program and a Grid service can handle many sub-jobs at a time. The performance evaluation showed that the proposed algorithm creates solution of equal or better quality than most existed applicable algorithms within a very short time period. Short execution time and high quality solution are the decisive factor for the applicability of the method in real environments, because the error recovery can be performed efficiently.

References

[1] T. Braun et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810 – 837, 2001.

[2] R. Buyya et al. Scheduling parameter sweep applications on global grids: A deadline and budget constrained cost-time optimisation algorithm. *Software: Practice and Experience (SPE), Wiley Press*, 35(5):491 – 512, 2005.

[3] E. Deelman. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1(1):25–39, 2003.

[4] Y. Gao et al. Adaptive grid job scheduling with genetic algorithms. *Future Gener. Comput. Syst.*, 21(1):151 – 161, 2005.

[5] M. Hovestadt. Scheduling in hpc resource management systems:queuing vs. planning. *Proc. 9th Workshop on JSSPP at GGF8, LNCS*, pages 1–20, 2003.

[6] R. Lovas et al. Support for complex grid applications: Integrated and portal solutions. *Proc. 2nd European Across Grids Conference*, May 2004.

[7] D. Quan et al. Mapping grid job flows to grid resources within sla context. *Proceedings of the European Grid Conference,(EGC 2005)*, 3470:1107–1116, 2005.

[8] D. Quan et al. On architecture for an sla-aware job flows in grid environments. *Journal of Interconnection Networks, World scientific computing*, pages 245–264, 2005.

[9] D. Quan et al. 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*, pages 287–292, 2005.

[10] D. Quan et al. Sla negotiation protocol for grid-based workflows. *Proceedings of the International Conference on High Performance Computing and Communications (HPPC-05), LNCS 3726*, pages 505–510, 2005.

[11] D. Quan et al. Error recovery mechanism for grid-based workflow within sla context. *Accepted by International Journal of High Performance Computing and Networking (IJHPCN)*, 2006.

[12] D. Quan et al. Mapping heavy communication workflows onto grid resources within sla context. *Proceedings of the International Conference of High Performance Computing and Communication (HPCC06)*, 2006.

[13] A. Sahai. Automated sla monitoring for web services. *DSOM 2002, LNCS*, 2506:28–41, 2002.

[14] D. P. Spooner et al. Local grid scheduling techniques using performance prediction. *IEEE Proc. Computers and Digital Techniques*, pages 87–96, May 2003.

[15] A. Sulistio et al. A time optimization algorithm for scheduling bag-of-task applications in auction-based proportional share systems. *Proceedings of the 17th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2005, IEEE CS Press)*, 2005.