New Aspect of Investigating Fault Sensitivity of Scientific Workflows

E. Kail[1], P. Kacsuk[2,3] and M. Kozlovszky[1,2]
[1] Óbuda University, John von Neumann Faculty of Informatics, Biotech Lab
Bécsi str. 96/b., H-1034, Budapest, Hungary, Country
[2] MTA SZTAKI, LPDS, Kende str. 13-17, H-1111, Budapest, Hungary
[3]University of Westminster, 115 New Cavendish Street, London W1W 6UW
{kail.eszter, kozlovszky.miklos}@nik.uni-obuda.hu,
kacsuk@sztaki.mta.hu

***Abstract* — Scientific workflows have emerged as a key technology that assists scientists with the design, management, execution, sharing and reuse of in silico experiments. These experiments are mainly data and compute intensive so they require high computing infrastructures to execute them. Since the complexity of these computing infrastructures is high so fault tolerant behavior must be supported to ensure the successful termination of the workflows while keeping to soft or hard deadlines and maintain usage costs at an optimum level. In this work we introduce a new aspect of supporting fault tolerance behavior or even task scheduling methods in time critical applications, which is based primarily on the topology of the workflow model. We show simple examples to prove the theoretical significance of our proposal.**

I. INTRODUCTION

Scientific workflows are used in almost every research field to describe and to simplify the abstraction of complex scientific experiments. A scientific workflow is composed of computational steps that are executed in sequential order or parallel wise determined by some kind of dependency factors. We call these computational steps tasks or jobs, which can be data intensive and complex computations. Mostly we differentiate data flow or control flow scientific workflows. While in the former one the data dependency determines the real execution path of the individual computational steps, in the latter one there is an explicit task or job precedence defined.

These scientific workflows can be represented by directed graphs $G\left(V,\vec{E}\right)$, where the nodes or vertices $v_i \epsilon V$ are the computational tasks or jobs and the edges between them represent the dependencies (data or control flow). Fig. 1 shows an example of a scientific workflow with 4 tasks (T_0, T_1, T_2, T_e), with T_0 being the entry task and T_e being the end task. The numbers assigned to the tasks represent the execution time that is needed to successfully terminate the task and the numbers assigned to the edges represent the time that is needed to submit the successor task after the predecessor task has been terminated. This latter one can be the data transfer time, resource allocation time or communication time between the consecutive tasks.

These tasks alone can be data and compute intensive and even time consuming computations thus may require high computing infrastructures (HPCs, grids and clouds) to execute them. In such a complex environment during the long execution time the environmental conditions are continuously changing due to system errors, network unavailability and user interventions and so on. Under such circumstances it is inevitable to use fault tolerant mechanisms to help successfully terminating the scientific experiments. Fault tolerance is the ability of a system to perform its functions even in the presence of a failure.

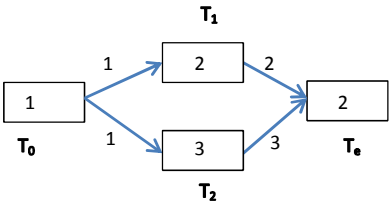

Figure 1. Scientific workflow with 4 tasks (T_0, T_1, T_2, T_e)

However, independently from the actual fault tolerant policy the method in itself has its own costs concerning execution time as well as resource usage, energy usage or even storage capacity.

Each workflow is bounded by a user defined deadline and also constraints may be specified between tasks as well. Hard deadline means, that results are useful only before the given deadline, while soft deadline means, that user may be unsatisfied with the quality of the service, but results are still useful [3].

In real-life the users (scientist) also may need to know the estimation of the execution time of their workflow before the submission. It has gained more importance nowadays with cloud systems where applications are delivered as services on a pay-per-use basis. Moreover, time critical applications need especially perfect timing. These factors lead to the need to minimize execution time and the effects of various failures occurring during execution time.

In one of our earlier work [2] we have defined the local and global cost of a failure occurring during the execution time of a task T_i and introduced a new approach of adapting the checkpointing interval based on the relation of the following two cost definitions. The local cost of a

failure on task T_i is the execution time overhead of task T_i when during execution one failure occurs. The global failure cost of a task T_i is the execution time overhead of the whole workflow, when one failure occurs during Task T_i.

Based on the above mentioned dependencies this work focuses on time critical applications and give a new aspect to minimize failure effects based on workflow structure analyses. We introduce fault sensitivity of a workflow topology and influenced zone of a task. According to these values the resource selection for the tasks can be optimized. For example if a task has an influenced zone consisting of a few tasks only then we can risk to schedule this task on a less reliable resource while those tasks that have the whole workflow as their influenced zone, in other words critical tasks should be executed on reliable resources. In general fault tolerance behavior can be adjusted based on the fault sensitivity property of a task.

The paper is organized as follows. In the next section we give a brief overview about connecting research areas. In chapter III we introduce definitions of influenced zone of a failure and sensitivity index of a graph and show the calculation methods on simple examples. We also show the theoretical significance and usage areas of our work. After a brief conclusion the bibliography closes our work.

II. Related Work

A. Fault Tolerance

The use of some kind of fault tolerance policy is essential regarding parallel and distributed infrastructures such as HPCs, grids and clouds. We differentiate reactive and proactive fault tolerant behaviour. While the aim of proactive techniques is to avoid situations caused by failures by predicting them and taking the necessary actions, reactive fault tolerance policies reduce the effect of failures on application execution when the failure effectively occurs. There are several solutions in the literature for fault tolerant behavior and other complementary methods in its connected fields [1]. However the most widely used methods are monitoring functions to notice the faults as soon as possible, checkpointing and resubmission, where the system state is captured and saved based on predefined parameters (i.e.: time interval, number of instructions) and when the system undergoes some kind of failure the last consistent state is restored and computation is restarted from that point on. Finally the replication where critical system components are duplicated using additional hardware or with scientific workflows critical tasks are replicated and executed on more than one processor [4]. The idea behind task replication is that replication size r can tolerate r-1 faults while keeping the impact on the execution time minimal.

Most of the already existing fault tolerance methods try to tackle the problem from the resource point of view. The authors of [5, 6, 7, 8] make some assumptions or gather statistics about failure distribution of individual resources or resource systems and based on these values and calculations adjust fault tolerant mechanisms. For example Theresa et al propose in their work [8] two dynamic checkpoint strategies: Last Failure time based Checkpoint Adaptation (LFCA) and Mean Failure time based Checkpoint Adaptation (MFCA) which takes into account the stability of the system and the probability of failure concerning the individual resources. Young in [5] has already in 1974 defined his formula for the optimum periodic checkpoint interval which is based on the checkpointing cost and the mean time between failures (MTBF) with the assumption that failure intervals follow an exponential distribution. Di et al in [6] has also derived a formula to compute the optimal number of checkpoints for jobs executed in the cloud. His formula is generic in a sense that it does not use any assumption on the failure probability distribution only the expected numbers of failures for a given task is considered.

The above mentioned fault sensitivity analyses focuses on resource reliability without considering the information that can be obtained from the workflow model or structure.

B. Workflow Structures

Investigating the structures of the workflow model is also a very important research field, and it is widely used in exception handling, scheduling mechanisms and also in workflow execution time estimation problems. All these problems may need to change the model by partitioning or clustering tasks. In [10] a performance prediction model is presented to estimate execution time of scientific workflows for a different number of resources, taking into account their structure as well as their system-dependent characteristics. Recently the most prevalent researches that investigate workflow structures are workflow similarities investigations [9] to improve the storing and sharing reproducible workflows.

However to the best of our knowledge there is no work dealing with workflow structures in order to improve fault tolerant behavior or task scheduling methods to increase reliability and thus the successful termination of time critical applications. In this work we take into account the rigidity or flexibility of the workflow, or in other words the fault sensitivity of the workflow model, which tells us what problem is caused if failure happens during the execution of a task. It is especially important concerning time-critical applications.

III. Fault sensitivity analyses

A workflow model is said to be sensitive if a failure occurring during the execution of a task (or a few occurrences of failures) causes the total workflow execution time to increase. To formulate the sensitivity of a workflow model we define the influenced zone of an individual task T_i: as the set of tasks which submission time are affected because a failure is occurred during the execution of task T_i. In other words if a failure does not have global effect on the workflow execution time, then we can define the border of its effect, or the set of tasks which submission occur at a later time due to this failure.

In this work we only consider workflow models where their graph representations are DAGs (Directed Acyclic Graphs) with one entry task T_0 and one end task T_e.

In such graphs from the entry tasks to the end tasks there can be several different paths. If there is only one path from T_0 to T_e i.e. the graph execution model is sequential than the sensitivity of the graph is very high, so very strict fault tolerance method should be used, because all failures have global effects.

If a task is part of all of the paths that exist from T_0 to T_e then it can be easily noticed that these tasks have high sensitivity and thus large influenced zones belong to them, because occurring a failure during the execution of these tasks the overall makespan is under all circumstances increased with the local cost of this failure, i.e.: in this case local cost is equal to global cost of these failures.

Those tasks that are not part of all paths from T_0 to T_e may have less influenced zones, those paths that do not contain this task may have longer execution path, so the failure cost will not affect the global makespan.

A. *Sensitivity index of the whole graph*

We define the sensitivity index (S) of a graph $G(V,\vec{E})$ as the ratio of the influenced zone to the remaining subgraph summarized by all tasks, and averaged over all tasks,

$$S = \frac{\sum_{i=1}^{|V|} \frac{|V_i|}{|V_{R,i}|}}{|V|} \quad , \qquad (1)$$

where V_i is the influenced zone of vertex v_i and $V_{R,i}$ is the remaining subgraph which can be formed by deleting the already terminated and submitted jobs from graph $G(V,\vec{E})$.

B. *Simple graphs*

In the next examples we worked with simple workflow graphs that consist of homogeneous vertices i.e. all the tasks have uniform properties and all the edges are also uniform, that is the communication costs, the resource allocation costs and network costs are considered to be identical for all task-pairs.

However it can be easily proved that an arbitrary graph model can be transformed to simple graphs that are built from homogeneous vertices and edges with steps that do not change the complexity of the workflow and do not change the connectivity property of the graph. In every transformation step a new vertex and a new edge is added in a form, that every new vertex is always connected to an already existed one with the new edge.

The workflow model in Fig. 1 contains numerical values assigned to the tasks and also to the edges. In our investigations these values represent estimated or retrieved time values from historical workflow runs that are needed to execute the actual tasks and to transfer data or allocate resource in order to submit the successor tasks. If we assign the value 1 to the uniform edges, than the original workflow in Fig. 1 can be transformed into the following structure in Fig. 2, where the tasks are partitioned and substituted with a path that length is equal to the execution time of the task and also the edges are substituted with paths that length are equal to the time that is needed to submit the successor task.

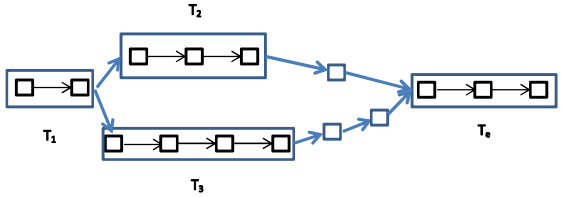

Figure 2. The transformed graph from Fig. 1

C. *Calculating the influenced zones*

If we consider the scientific workflow model in Fig. 1 we can easily notice that task T_1 can be submitted 1 time unit after the termination of task T_0. It stands for T_2 as well, and T_e can only be submitted 3 time units after the termination of T_2. T_e receives the data from T_1 2 time units after the termination of T_1 but it has to wait for all its input data i.e. a task cannot be started until it has all its input data from all of its predecessors.

So it can be noticed that path (T_0, T_1, T_e) is shorter than path (T_0, T_2, T_e). If the difference between the length of these two paths is more than the time needed for a fault recovery during T_1 than we can say that the failure effects has been localized, the influenced zone of this task is only itself. While a failure occurring during T_2 would mean that T_e would be submitted also later, so the workflow would terminate later.

Calculating the influenced zone on graphs with uniform vertices and edges is much easier, because the time values can be obtained from merely the topology. We should only find the circles (neglecting the direction of the edges) that include task T_i and compare the two directed paths of the circle. In general the tasks on the longer path have global effects while the task on the shorter path may have less impact on the workflow execution time.

D. *Examples*

The first example is a very simple graph model containing 3 tasks (T_0, T_1, T_e) in sequential order (Fig. 3). As it was mentioned already in the previous section if a graph consists of only one path then all the failures occurring during the whole execution time has global effect.

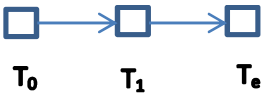

Figure 4. Scientific workflow with 3 tasks (T_0, T_1, T_e)

So the influenced zones of the tasks are the remaining subgraph starting from the actual task. In this case the sensitivity index of this graph can be calculated as follows:

$$S = \frac{\frac{3}{3} + \frac{2}{2} + \frac{1}{1}}{3} = 1 \qquad (2)$$

The second example Fig. 4 contains 5 tasks (T_0, T_1, T_2, T_3, T_e) and 2 different paths from T_0 to T_e, which means the graph includes one circle if we neglect the orientation of the edges.

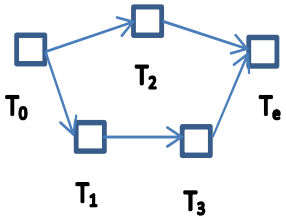

Figure 3. Scientific workflow with 5 tasks (T_0, T_1, T_2, T_3, T_e)

In this case the sensitivity index of this graph is calculated as follows (3): If a failure occurs during the execution of T_0 then it has effect on the whole graph i.e. it has effect on all the tasks. It is generally true for all workflows with one entry task. If the failure occurs during the execution of task T_1 then it has not effect on the predecessor task and on the tasks that are part of a path that does not include T_1 only on itself and its successors. So the other path of the circle is not affected, and even if the execution time of this path is shorter than the other path then the vertex that joins the two paths is also not affected, because a task cannot be submitted before all input data has not arrived from all of its predecessor tasks.

$$S = \frac{\frac{5}{5} + \frac{3}{4} + \frac{1}{4} + \frac{2}{2} + \frac{1}{1}}{5} = \frac{4}{5} \qquad (3)$$

Taking into account that all vertices and edges are homogeneous, the path execution time can be measured by the length of it. In this example the lower path is longer so if a failure occurs during T_2 then we can declare that it has only local affect, the recovery from the failure does not add extra latency to task T_e submission time and thus the whole workflow execution time.

Comparing these two examples the sensitivity indexes tell us that the first model is extremely sensitive while the second one has a minimal flexibility. It means that in the second case concerning task T_2 we can use less strict fault tolerance technique, because the system can tolerate some failures, and thus the overall cost (storage, bandwidth) of the used fault tolerance technique can be decreased. This value can also be taken into account when selecting the appropriate resource for a given task. Critical tasks which failure affect the total execution time of the workflow should be submitted to high reliability resources while those which failures has less impact can be executed on low reliability resources.

Finally, let us consider a bit more complex workflow Fig. 5. This model contains 4 circles, and it can be also noticed that there are tasks that are part of more than one circle. In this case at first we should find all the circles that include the task and then find the minimal size of them which satisfy the condition that the length of the two paths forming the undirected circles has a minimal difference that is equal or bigger then the time needed to recover from a faulty situation during that task.

In Fig. 5 in the circle with continuous line both paths have equal length, while both in the dashed and the dotted circles the filled squares represent those tasks that are part of the shorter path, so the influenced zone of these tasks can only be bounded to the path that they belong to.

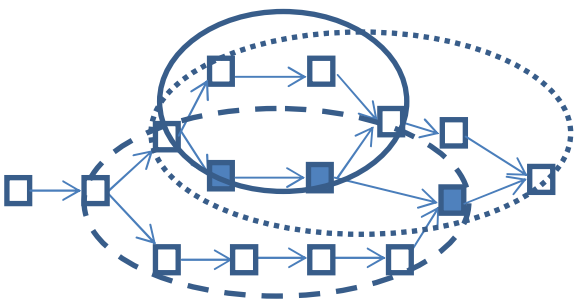

Figure 5. Workflow graph with circles and influenced zones

E. *Theoretical significance*

It can be seen that on simple examples it is very easy to calculate the influenced zones and the sensitivity index, but in more complex workflow structures with high number of vertices it would need very long time to carry out an exhaustive search for all tasks. As the number of the vertices and the complexity of the workflow model increases the calculation steps can increase exponentially. So the results of this method only show us the theoretical possibility to take into account the workflow structures to adjust fault tolerant methods or scheduling tasks.

IV. CONCLUSION

In this paper we introduced a new aspect of analyzing fault sensitivity. Instead of calculating the failure probability of the different high computing infrastructures such as HPCs, grids and clouds we investigated workflow structures in order to predict the impact of the faults on the whole execution time of the workflow. We defined sensitivity index and showed the calculation method on simple examples. We pointed out that the usage of these calculations has only theoretical significance since as the number of the vertices and the complexity of the workflow model increases the calculation steps can increase exponentially. As a part of our ongoing work we would like to find a polynomial time algorithms that has also practical usage and then adapt the algorithm to dynamically changing workflow models as well.

REFERENCES

[1] Bala, Anju, and Inderveer Chana. „Fault tolerance-challenges, techniques and implementation in cloud computing". IJCSI International Journal of Computer Science Issues vol. 9, 2012

[2] E. Kail, P. Kacsuk, M. Kozlovszky: "Achieving dynamic workflow management system by applying provenance based checkpointing method", MIPRO, 2014 Proceedings of the 38th International Convention

[3] Failure prediction for scalable checkpoints in Scientific Workflows Using Replication and resubmission task in Cloud Computing. Palaniammal, Santhosh

[4] Laiping Zhao; Yizhi Ren; Yang Xiang; Sakurai, K., "Fault-tolerant scheduling with dynamic number of replicas in heterogeneous systems," High Performance Computing and Communications (HPCC), 2010 12th IEEE International Conference on, vol., no., pp.434,441, 1-3 Sept. 2010

[5] J.W. Young. "A first order approximation to the optimum checkpoint interval" in Communications ACM, 1974

[6] S. Di, Y. Robert, F. Vivien, D. Kondo, Cho-Li Wang, and F. Cappello. „Optimization of Cloud Task Processing with Checkpoint-Restart Mechanism", 1–12. ACM Press, 2013. doi:10.1145/2503210.2503217.

[7] B. Meroufel, G. Belalem: "Adaptive time-based coordinated checkpointing for cloud computing workfl ows". in Scalable Computing: Practice and Experience, Vol 15, No 2, 2014

[8] Antony Therasa.S, Sumathi. G, Antony Dalya.S.: "Dynamic Adaptation of Checkpoints and Rescheduling in Grid Computing" in International Journal of Computer Applications vol. 3, 2010A

[9] Starlinger, J. ; Cohen-Boulakia, S. ; Khanna, S. ; Davidson, S.B. et al.: "Layer Decomposition: An Effective Structure-based Approach for Scientific Workflow Similarity", 2014 IEEE 10th International Conference on e-Science, vol.1, pp.169,176, 20-24 Oct. 2014, doi: 10.1109/eScience.2014.19

[10] I. Pietri, G. Juve, E. Deelman, R. Sakellariou: "Performance Model to Estimate Execution Time of Scientific Workflows on the Cloud" in Proceedings of WORKS'14