Dynamic Scheduling of Multi-Processor Tasks on Clusters of Clusters

Sascha Hunold [#], Thomas Rauber [#], Gudula Rünger [*]

[#]*Department of Mathematics and Physics*
University of Bayreuth, Germany
{hunold,rauber}@uni-bayreuth.de

[*]*Department of Computer Science*
Chemnitz University of Technology, Germany
ruenger@informatik.tu-chemnitz.de

***Abstract*— In this article we tackle the problem of scheduling a dynamically generated DAG of multi-processor tasks (M-tasks). At first, we outline the need of such a scheduling approach in the context of TGrid. TGrid is an M-task runtime system for heterogeneous clusters. Then, we propose a dynamic scheduling algorithm called Reuse Processors Algorithm (RePA). The main objective of RePA is to reduce the communication and redistribution costs by trying to map child tasks to processors which are assigned to parent tasks (reuse processors). The algorithm is implemented using the SimGrid toolkit and is evaluated by comparing the makespan of the schedules produced by RePA and M-HEFT.**

I. Introduction

PC clusters have become mainstream for running very time-consuming parallelized applications. Nowadays, there are many installations of those PC clusters available to developers and researchers. The computational power of the parallel platform is the limiting factor for the quality (time, detail) of the result of a simulation application. An efficient utilization of multiple clusters is the key to improve the quality of the results significantly.

One kind of parallel applications is the mixed-parallel application which often leads to faster execution times than pure data-parallel applications [1]. A mixed-parallel application consists of tasks which can be executed concurrently and these tasks can be implemented in a data-parallel way. A task which is assigned to a number of available processors is called a multi-processor tasks (M-tasks). A mixed-parallel program which consists of M-tasks can be represented by a direct acyclic graph (DAG) where the edges denote data dependencies between the M-tasks. If a task B requires data from a task A to become executable then B depends on A which is indicated by a data dependency edge. In most cases, these task-graphs are directed acyclic graphs (DAGs).

A lot of research has been done on scheduling this kind of mixed-parallel applications (DAGs) on homogeneous platforms [2], [3]. More recently, heterogenous platforms (grids, clusters of clusters) have become the target for scheduling mixed-parallel applications [4], [5]. Previous work addresses the problem of scheduling DAGs for which all nodes and edges are known before the scheduling process starts. However, the execution graph of mixed-parallel application can also be created dynamically at runtime. For instance, in a recursive process a task may create new sub-tasks if a certain criterion is satisfied. Thus, the entire DAG is not known beforehand.

Scheduling such dynamically generated DAGs in heterogenous environments (a collection of clusters) is a challenging task. Assigning too many processors to one task may prevent other tasks from being executed. On the other hand, assigning a lot of processors does not necessarily reduce the execution time of a task by a large amount since the runtime is limited by the fraction of the task which is executed sequentially. It is therefore an important criterion for a scheduling algorithm to produce schedules with a small makespan. The makespan is the time required to execute all the nodes in the DAG. The scheduler also has to pay attention to the resulting communication costs. In general, a task receives input data, performs some computation, and then produces output data. Input and output data (e.g. matrices or vectors) are distributed across the executing processors of one task. Data redistribution is required in order to use data from a source task (parent) in a target task (child), e.g. a block-partitioned matrix shared by five processors is used as input data for a task which is executed by only three processors. Depending on which processors are assigned to source and target tasks, the communication costs vary tremendously, especially when the tasks are mapped to different clusters and the clusters are connected over a low bandwidth link.

In previous work we have introduced the TGrid framework which is an execution environment for multi-processor tasks on clusters of clusters [6]. It supports the dynamic creation of M-tasks, leading to aforementioned dynamically created DAGs. M-task programs can be used to implement scientific applications which are formulated recursively, e.g. the generation of new M-tasks is stopped when a pre-defined convergence criteria is met. New tasks can be created and already created tasks can become executable if all data dependencies are satisfied. Whenever tasks are available the TGrid scheduler has to decide which processors should be assigned to these tasks.

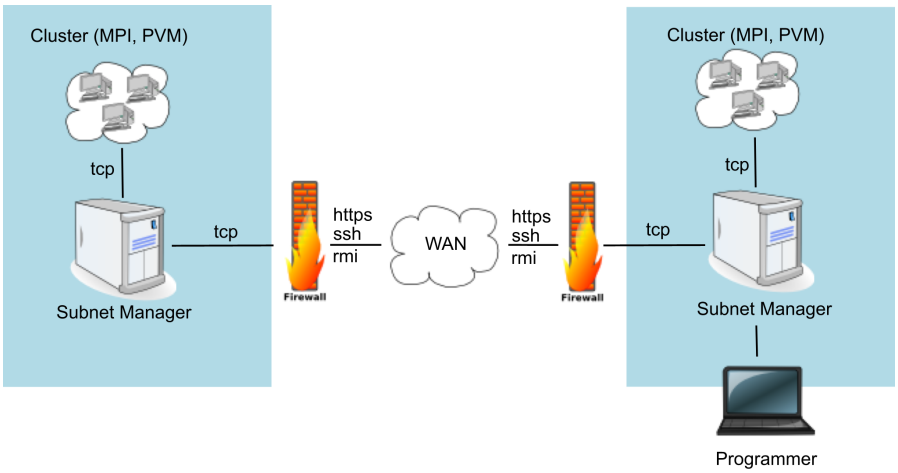

Fig. 1. TGrid architecture

The contribution of this article is the proposal of a novel algorithm for scheduling dynamically created DAGs of M-tasks onto clusters of clusters. In the context of this work the term *dynamic* is used to refer to DAGs which are dynamically generated. It does not imply that a single M-task can be dynamically rescheduled once it is running. The herein introduced algorithm RePA generates competitive schedules (often with a smaller makespan) in comparison to recently introduced static scheduling algorithms for M-tasks such as M-HEFT [4]. Moreover, the algorithm has a small computational complexity, i.e. it produces a mapping of ready tasks onto clusters in a short amount of time which makes it suitable to be used inside the online scheduling component of the TGrid framework. The performance of RePA is evaluated by a detailed investigation of the resulting makespan for a large number of DAGs in different grid environments.

This paper is organized as follows. Section II gives an overview of the TGrid environment. Section III introduces the ReP algorithm for the dynamic scheduling of M-tasks. Section IV presents the experimental evaluation of the ReP algorithm. Section V discusses related work and Section VI concludes the article and outlines future work.

II. TGrid: Multi-processor task programming on the grid

This section summarizes previous work and background information on TGrid. In the second part, we define the model and state the assumptions for the scheduling problem.

A. The TGrid environment

TGrid is a software runtime system which allows the execution of hierarchically-structured multi-processor tasks [6]. M-tasks which are bound to disjoint sets of processors can run concurrently and thus implement the task parallel paradigm. M-tasks can also be implemented in a data-parallel way, e.g. with MPI. M-task programs can have a nested structure, i.e. M-task can be built from other M-tasks or can be formulated recursively. In general, the mixed-parallel paradigm increases the degree of parallelism of an application. The use of M-tasks may also reduce the communication overhead through exploiting spatial data locality.

TGrid is the glue to combine clusters into a bigger cluster of clusters and to run M-tasks programs on these bigger clusters. A sketch of the TGrid architecture is depicted in Figure 1. Each cluster is controlled by a subnet manager which is in charge of executing and observing tasks on its private network. Such a private network could be a homogeneous cluster or any other heterogenous collection of machines that share a private IP address space. The subnet managers have to transfer data through a possibly insecure WAN. Therefore, the subnet managers support several protocols, such as https, ssh, etc., in order to bypass local firewalls and to perform encrypted data transfer.

A program developer has access to one of the subnet managers and can submit programs to a local subnet manager. A TGrid program can be represented as DAG where TGrid tasks are represented by nodes. Dependencies between tasks are modeled as edges. Data dependencies are automatically resolved by TGrid [7], i.e. the redistribution component queries the sending (preceeding) and receiving (succeeding) task for the format of input/output data (size, type, etc.) and performs the data redistribution (coupling of tasks).

TGrid and all its components are written in Java which makes them completely platform-independent. The current implementation supports M-tasks which are implemented using MPI. This approach allows us to run these components on arbitrary machines in the grid without having to recompile some parts. Moreover, the ability to access the MPI layer through JavaMPI enables the application to benefit from using proprietary network drivers (Infiniband) on the target machine.

B. Scheduling problem for M-tasks

As described above, a TGrid subnet is controlled by a manager daemon. This subnet manager is not only responsible for receiving M-task programs from the programmer, it also contains a scheduling component which maps runnable M-

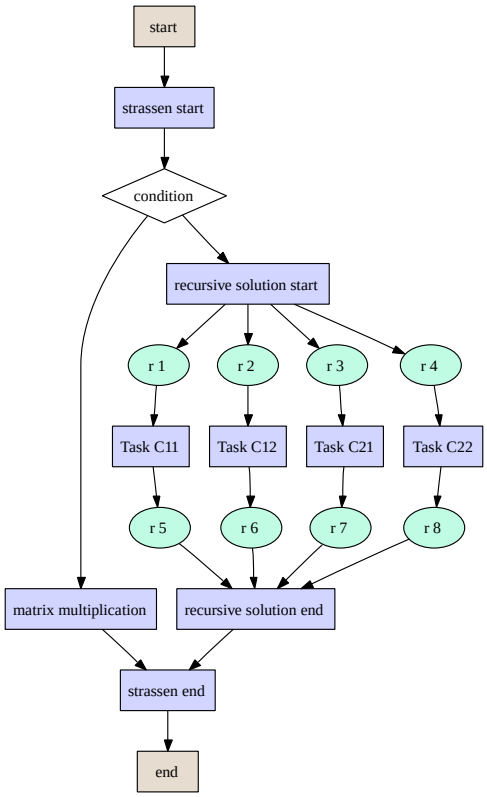

Fig. 2. DAG showing a mixed-parallel Strassen matrix multiplication. Rectangles represent computation nodes. Elliptic nodes represent data redistribution tasks. The nodes *Task Cij* (i,j = {1,2}) are recursively structured M-tasks, i.e., internally they consist of all nodes from *strassen start* to *strassen end*.

tasks onto one of the participating clusters. In the remaining article we tackle the problem of designing a scheduling algorithm in order to implement the TGrid scheduling component.

Even though TGrid has been designed to support heterogeneity within a single cluster (a cluster may consist of a heterogeneous set of processors), in this article we only consider clusters of homogeneous clusters. This is motivated by the fact that most compute clusters consist of a homogeneous collection of machines. Thus, we assume a computational platform that consists of c clusters C_i, where each cluster C_i contains p_i processors of the same type and speed. All clusters are connected to a backbone by a single network link.

Note that the DAG in the context of TGrid refers to the final DAG after executing all tasks of a M-task program. A sample DAG for a mixed-parallel implementation of Strassen's algorithm is shown in Figure 2. This sample DAG comprises an artifical start and end node which are only markers used by the scheduler. The blue rectangles denote computation nodes and the green ellipses denote data dependencies between computation nodes. All computation nodes have a specific computation cost (number of instructions) when being processed. Similarly, redistribution nodes have costs assigned to them, e.g. the number of bytes to transfer. For the sake of completeness, our model allows multiple redistribution nodes between pairs of computation nodes.

III. The ReP algorithm

In this section, we introduce the ReP algorithm for scheduling dynamically created DAGs of M-tasks. As described above, the scheduler of the TGrid framework should be able to make good decisions about mapping ready tasks onto clusters. A task becomes ready when all data dependencies are satisfied, i.e. all parent nodes have been executed. A good dynamic scheduler has to match the following conditions. On the one hand, the decision making process has to be done in a small amount of time. On the other hand, the algorithm has to create a schedule leading to a small makespan and good parallel efficiency. We make a few assumptions for designing the scheduling algorithm. It is assumed that tasks are always mapped to a single cluster. Otherwise meta data and cost functions of the internal task structure are required to make predictions of the computation time and the communication overhead inherited by assigning a task to more than one cluster. The reason is that finding a good cost function is fairly complex and the prediction precision is also questionable. Moreover, most of the data parallel code is not performing well in strongly heterogeneous distributed environments mostly due to a low latency between clusters.

The overall performance of M-task programs which are executed on TGrid is primarily bounded by the communication costs inherited when child tasks are mapped to a different set of processors or to a remote cluster. It is assumed that there is

Algorithm 1 REPA

```
1: INPUT: queue - ready nodes
2: sort queue by decreasing computation cost
3: sort queue by increasing node depth (stable sort)
4: while queue is not empty and at least one cluster is available do
5:    node = queue.pop()
6:    cluster = find_target_cluster(node)
7:    processor_nb = get_number_of_processors(node, cluster)
8:    processor_list = select_processors(node, cluster, processor_nb)
9:    add new schedule(node, cluster, processor_list)
10: end while
```

function find_target_cluster(node)

```
1: return cluster with most available computational power
```

function get_number_of_processors(node, cluster)

```
1: if node is root node then
2:    return number of free processors on cluster
3: else
4:    free_processor_nb = number of free processors on cluster
5:    cluster_power_ratio = (free computational power of cluster) / (free computational power of all clusters)
6:    node_computation_ratio = (computation cost node) / (computation cost of unscheduled ready nodes)
7:    return max( (min(1, node_computation_ratio / cluster_power_ratio ) * free_processor_nb), 1 ) // at least one processor, at most the whole cluster
8: end if
```

function select_processors(node, cluster, processor_nb)

```
1: processor_list = empty list
   // try to reuse processors which were assigned to any parent node
2: append all free processors to processor_list which are assigned to a parent node on cluster
3: append all other free processors on cluster to processor_list
4: return first processor_nb processors of processor_list
```

no communication cost for those processors which are shared between the parent and the child task. Thus, the ReP algorithm tries to *reuse* as many parent processors as possible for a given task in order to reduce communication costs. Yet, there are limitations for assigning parent processors to child tasks, e.g. if a parent task has more than one child the parent processors have to be distributed among the children. The ReP algorithm addresses these problems and tries to make fair decisions when assigning processors to child tasks.

The pseudo-code of the ReP algorithm is presented in Algorithm 1. The first 10 lines of code are always executed when new nodes (tasks) become ready. The ready nodes are first ordered by the computation cost and the length of the path from the root node (depth). The node depth is used to enforce scheduling in breadth-first search (BFS) style. This avoids following a single path in the DAG which could prevent many nodes from becoming executable. The ordered list of ready nodes is then processed until all nodes have been mapped on a cluster or there is no cluster with idle processors left. The assignment process is done in three steps. In step (1), the algorithm determines the target cluster for a node by function `find_target_cluster`. Then, in step (2), the scheduler decides how many processors should be assigned to this node using function `get_number_of_processors` and in step (3), the scheduler selects this number of processors from the target cluster by `select_processors`. RePA uses a heuristic approach in each step. It turned out that it is efficient to select the cluster with the most free computational power. The function `get_number_of_processors` computes the number of processors for a node on a target cluster. This function is used to assign a number of processors which is proportional to a node's computational cost. Since a task is placed on exactly one cluster, we also consider the relative computational power of the target cluster which also gives sibling nodes a chance to reuse parent processors. In the last step, p processors of the target cluster are selected. Only at this point, the reuse of processors comes into play. The algorithm will first assign as many processors of parent nodes as there are available. When there is no idle parent processor left the scheduler continues to assign processors from the list of remaining idle processors.

The ReP algorithm as it is shown in Algorithm 1 is a result of experimenting with several heuristics for choosing the target cluster and for selecting the number of processors. We noticed that the most critical step for obtaining a small makespan is step (1), the selection of the target cluster. We have experimented with different heuristics for determining the target cluster that give clusters with more parent processors a higher priority than clusters with less or no parent processors. This might be a reasonable solution for nodes with high communication costs and small computational costs. However, our sample DAGs do not follow this pattern. Thus, favoring clusters with more parent processors before clusters with more free computational power in the cluster selection step (1)

does not lead to small makespan. In fact, in this case RePA tends to use only a small number of clusters (those which have been assigned to tasks in the starting phase), so that efficient utilization of all participating clusters is not achieved. Therefore, we decided to consider the *parent processor reuse* only during the processor selection step (3).

IV. Experimental evaluation

In this section we want to quantify the quality of the dynamic scheduling approach by comparing the makespan of schedules generated by RePA and M-HEFT [4].

A. Experimental setup

The experiments have been performed using SimGrid toolkit [8] which provides a testbed containing all necessary functionalities to run a simulation of mixed-parallel programs in a heterogeneous distributed environment. Let us recall, that the heterogenous computation platform in this article is based on a collection of homogenous clusters. All clusters are connected to a single backbone and it is assumed that the interconnection-network of clusters is fully connected. We use the same inter-connection network for each cluster with a bandwidth of 1 Gbit/s.

As described in Section II-B a computation tasks has associated computation costs given in Flop. To run such a task in SimGrid, we have to specify the computation time of this task. The model used for obtaining this estimation is based on Amdahl's law

$$T(t,p) = \left(\alpha + \frac{1-\alpha}{p}\right) \cdot \tau \ .$$

Amdahl's law states that the computation time of an M-task t is the accumulated time spent in the non-parallelizable fraction of the program (α) as well as in the parallelizable fraction ($1-\alpha$), where p is the number of processors and τ is sequential time of task t. We refer to [3] for a good overview of runtime prediction using Amdahl's law for data-parallel tasks. Consequently, the nodes of the sample DAGs for the simulation have an attribute specifying the α value in Amdahl's law. For the experiments, the α value was randomly chosen between 0 and 0.2.

For evaluating the performance of RePA, we generated a number of DAGs and grid descriptions. We created 50 random DAGs consisting of 50, 75, and 100 computation nodes respectively. Due to the time complexity of M-HEFT we did not compare the scheduling performance of larger DAGs.

The shape of the DAGs can be controlled by the value fat which is intended to specify the width of the DAG. Thus, the fat models the degree of parallelism of the DAG, i.e. the bigger the width the more tasks can be executed in parallel. It denotes the percentage of independent tasks in the DAG, e.g. a fat value of 0 results in a single chain of nodes, and for $fat = 1.0$ all nodes in the DAG are independent. In the experiments, we have used DAGs with a fat value of 0.6 and 0.9.

All DAGs are randomly generated and the minimum execution cost of each task was set to 10.000 Flop. For more information about the dag generation program see [9].

Similar to the DAGs, we also generated several grid configurations for the experiments. We created two different types of grids. The first contains a homogeneous collection of clusters whereas the second is heterogenous. In the homogeneous case, all processors across all clusters have the same processor type, but clusters may consist of a different number of processors. In either case (homogeneous and heterogeneous) the number of processors per cluster is randomly chosen between 16 and 64 which is a common setup of many cluster installations. We generated 10 grid setups consisting of 4 and 8 clusters, respectively. In total, 40 grid configurations were generated, 10 configurations with 4 (8) homogeneous clusters, and 10 with 4 (8) heterogenous clusters.

To evaluate the quality of RePA we decided to compare the makespan of the schedules produced by RePA and M-HEFT [4]. The reasons for choosing M-HEFT are as follows. First, it was one of the first algorithms that directly addressed the problem of scheduling mixed-parallel programs in heterogenous distributed environments showing good performance. Secondly, to obtain comparable results we had to use the same testing framework and we had access to an implementation of M-HEFT for SimGrid.

As discussed above, the ReP algorithm does not consider the redistribution costs between tasks in the decision making process. However, the time spent in data redistribution is simulated within the SimGrid framework. The time prediction function for the data redistribution time is based on a full block distribution of data which is used in ScaLAPACK [10].

B. Introduction to M-HEFT

M-HEFT [4] is based on the HEFT algorithm [11]. The HEFT algorithm is an extended list scheduling algorithm for heterogeneous environments and has three proceeding stages. At first, the mean values of the computation costs of nodes and communication costs of edges are computed. In the next step the algorithm assigns the *upward rank* to each node. This rank basically determines the critical path from a node (task) to the exit node. The rank is recursively defined starting with the exit node. According to this rank, the list of nodes is sorted in decreasing order. In the last step, the task with the highest priority is mapped to the target platform until all tasks are scheduled. Each task is scheduled using the processor allocation that minimizes the finish time.

The M-HEFT algorithm extends HEFT by considering the data-parallel execution of tasks. Similar to RePA, the target platform of M-HEFT is a heterogenous collection of homogeneous clusters. The original communication model of HEFT had to be adapted to support data-parallel tasks. Thus, additionally to serial communication times M-HEFT also considers the redistribution costs between processor configurations of two dependent tasks.

TABLE I

RELATIVE PERFORMANCE OF REPA COMPARED TO M-HEFT FOR 4 CLUSTERS

50 nodes		homogeneous	heterogenous
min RePA makespan	fat=0.6	88.2%	87.7%
	fat=0.9	81.2%	82.0%
max RePA makespan	fat=0.6	136.7%	152.6%
	fat=0.9	124.8%	145.0%
avg RePA makespan	fat=0.6	106.7%	113.8%
	fat=0.9	103.7%	109.7%
RePA wins / #tests	fat=0.6	96/500	48/500
	fat=0.9	181/500	111/500
75 nodes		homogeneous	heterogenous
min RePA makespan	fat=0.6	84.0%	88.5%
	fat=0.9	72.3%	72.5%
max RePA makespan	fat=0.6	139.6%	159.4%
	fat=0.9	117.1%	132.8%
avg RePA makespan	fat=0.6	108.6%	117.8%
	fat=0.9	98.9%	108.8%
RePA wins / #tests	fat=0.6	80/500	19/500
	fat=0.9	270/500	103/500
100 nodes		homogeneous	heterogenous
min RePA makespan	fat=0.6	91.0%	94.9%
	fat=0.9	78.3%	83.9%
max RePA makespan	fat=0.6	131.4%	147.4%
	fat=0.9	117.3%	126.6%
avg RePA makespan	fat=0.6	109.4%	118.2%
	fat=0.9	98.7%	106.6%
RePA wins / #tests	fat=0.6	51/500	9/500
	fat=0.9	277/500	95/500

C. Results

A summary of the experimental results obtained for 4 and 8 clusters are given in Table I and Table II. The tables are divided into three sections, where each section contains the results for DAGs with either 50, 75, or 100 nodes. For each number of nodes, several performance measures are presented. There are three different values of the makespan. Scheduling a DAG D_i on a grid G_j using RePA (M-Heft) results in a makespan $M_{REPA}(i,j)$ ($M_{MHEFT}(i,j)$). The relative makespan of RePA is defined as

$$M_{rel}(i,j) = \frac{M_{REPA}(i,j)}{M_{MHEFT}(i,j)} \cdot 100.$$

The minimum makespan denotes the smallest relative makespan

$$M_{rel}^{min} = \min_{\forall D_i \forall G_j} \{M_{rel}(i,j)\}.$$

The maximum and the average makespan are computed accordingly. The last line in each section shows how many times RePA has created a schedule with a smaller makespan than M-HEFT. The value "#tests" denotes the total number of tests which have been done for this configuration (50 DAGs tested on 10 different grids). The value fat indicates the degree of parallelism (width) of the DAGs. The results are given for the homogenous and the heterogeneous configurations.

We can observe from the data in Table I that the average makespan decreases with an increasing number of nodes. Thus, the quality of the schedules produces by RePA increases with larger DAGs. For scheduling highly parallel DAGs (fat=0.9) containing 75 and 100 nodes on a homogeneous set of cluster, RePA produces schedules with an average makespan of less than 100% and RePA wins more than 50% of all tests (270 and 277). Thus, we can state that RePA outperforms M-HEFT in these cases. For DAGs with a smaller degree of parallelism (fat=0.6) M-HEFT leads to a better average makespan. However, RePA wins up to 19% of the tests (50 nodes, homogeneous).

In the heterogeneous case, the RePA is only slightly less effective than for the homogeneous configuration. The average makespan for 50, 75, and 100 nodes is 109%, 108%, and 106%, respectively. These results are still very good considering the disadvantages of the dynamic scheduler. First, the scheduler knows nothing about the implication of the current scheduling decision (entire DAG unknown). This directly implies another disadvantage. Since M-HEFT has already made a decision of where to place all computation nodes, data redistributions can already be started when data of any parent node are available even if the child node is not yet executable. This is not possible in a dynamic approach. The final assignment of processors to a task is only done when the task becomes executable and only from this time on data redistributions can be performed. Another factor with an impact on the performance of the dynamic scheduler is the information propagation. In our dynamic model, the scheduler runs on a dedicated processor on one of the clusters. When an M-task has been scheduled to another cluster the scheduler has to send a message to this target cluster requesting the execution of a task. Likewise, the target cluster has to notify the scheduler when the M-task has been completed. This adds to the total communication costs of RePA. Due to these

TABLE II
RELATIVE PERFORMANCE OF REPA COMPARED TO M-HEFT FOR 8 CLUSTERS

50 nodes		homogeneous	heterogenous
min RePA makespan	fat=0.6	104.3%	104.6%
	fat=0.9	85.3%	78.2%
max RePA makespan	fat=0.6	114.1%	150.0%
	fat=0.9	142.0%	169.9%
avg RePA makespan	fat=0.6	107.4%	121.7%
	fat=0.9	113.8%	115.0%
RePA wins / #tests	fat=0.6	0/500	0/500
	fat=0.9	59/500	71/500
75 nodes		homogeneous	heterogenous
min RePA makespan	fat=0.6	108.6%	104.5%
	fat=0.9	76.5%	73.9%
max RePA makespan	fat=0.6	132.6%	175.6%
	fat=0.9	135.7%	154.6%
avg RePA makespan	fat=0.6	116.6%	132.4%
	fat=0.9	108.9%	112.9%
RePA wins / #tests	fat=0.6	0/500	0/500
	fat=0.9	90/500	84/500
100 nodes		homogeneous	heterogenous
min RePA makespan	fat=0.6	110.1%	108.5%
	fat=0.9	81.2%	74.2%
max RePA makespan	fat=0.6	148.1%	172.4%
	fat=0.9	130.6%	145.5%
avg RePA makespan	fat=0.6	125.6%	137.1%
	fat=0.9	105.3%	101.9%
RePA wins / #tests	fat=0.6	0/500	0/500
	fat=0.9	153/500	270/500

message, the total makespan of RePA will always be bigger than the makespan of M-HEFT even if M-HEFT and RePA generate the same schedule (assign the same processors to each node). This also explains why RePA does not win more tests even if the average makespan is close to 100%.

The performance results of RePA for scheduling the same 50 randomly generated DAGs onto 8 clusters are given in Table II. We can make similar observations as for 4 clusters. The average makespan decreases with an increasing number of nodes and an increasing width of the DAGs (fat). However, we can see that RePA does not lead to a smaller makespan if the DAG is less wide ($fat = 0.6$). This is mainly reasoned by the extra communication between the scheduling host and the target clusters as described above. For a fat value of 0.6 and 8 clusters, the number of possible task placements is very limited. Thus, M-HEFT can take advantage of performing redistributions in advance and it avoids the extra messages between scheduler and target cluster. However, if the width of the DAG is larger ($fat = 0.9$) the performance of the schedules produced by RePA improves significantly. The reason is that M-HEFT "tends to use very large processor allocations for data-parallel tasks" [5] and therefore limits the number of concurrently running tasks.

The experimental results of our approach for dynamically scheduling M-tasks onto a set of clusters look promising taking into account that the scheduler has less information about the DAG than a static approach and has therefore less possibilities for making accurate predictions.

V. RELATED WORK

Existing algorithms for scheduling mixed-parallel programs are based on static DAGs. A number of scheduling algorithms have been designed for the case of homogeneous platforms, e.g. the algorithms CPR and CPA [2], [3].

Mixed-parallel task scheduling for more heterogenous environments have recently been developed, e.g HCPA and M-HEFT. A good overview of these algorithms as well as a discussion of modifications of M-HEFT to gain better performance are summarized in [5].

An algorithm for dynamic scheduling of directed graph-based workflows has been presented in [12]. Even though the workflow model used in this paper is similar to the M-task graphs of TGrid, the main objective of both approaches is different. Instead of scheduling a static workflow representation on a highly heterogeneous and dynamic grid environment, the TGrid program itself can be treated as a dynamic description of a mixed-parallel workflow which is executed in a static environment consisting of several homogeneous clusters.

VI. CONCLUSIONS

In this article we have presented a new algorithm for dynamic scheduling of DAGs of M-tasks onto clusters of clusters. The main objective of our scheduling approach is to keep the redistribution costs between subsequent tasks small by reusing processors. We have presented an evaluation of the scheduling algorithm by comparing the makespan of RePA and M-HEFT for a large number of DAGs and grid configurations. The experiments show good results of the dynamic scheduling approach considering that the entire DAG is unknown to

the algorithm and therefore the algorithm cannot make any predictions about placing succeeding tasks.

In future work, we will improve the heuristics for task migration by considering the redistribution costs when selecting the target cluster as proposed in [13]. Another goal is to implement this algorithm in the TGrid scheduling component and to evaluate the performance in a real world scenario.

ACKNOWLEDGMENTS

We would like to thank Emmanuel Jeannot, Frédéric Suter, and Tchimou N'Takpé of INRIA Lorraine for helpful discussions in support of this work and for giving us the access to the SimGrid scheduling framework for an evaluation of RePA.

REFERENCES

[1] S. Chakrabarti, K. Yelick, and J. Demmel, "Models and Scheduling Algorithms for Mixed Data and Task Parallel Programs," *J. Parallel Distrib. Comput.*, vol. 47, no. 2, pp. 168–184, 1997.

[2] A. Radulescu, C. Nicolescu, A. J. C. van Gemund, and P. Jonker, "CPR: Mixed Task and Data Parallel Scheduling for Distributed Systems." in *Proceedings of the 15th International Parallel & Distributed Processing Symposium (IPDPS-01)*. IEEE Computer Society, 2001, p. 39.

[3] A. Radulescu and A. J. C. van Gemund, "A Low-Cost Approach towards Mixed Task and Data Parallel Scheduling," in *ICPP '02: Proceedings of the 2001 International Conference on Parallel Processing*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 69–76.

[4] H. Casanova, F. Desprez, and F. Suter, "From Heterogeneous Task Scheduling to Heterogeneous Mixed Parallel Scheduling," in *Proceedings of the 10th International Euro-Par Conference (Euro-Par'04)*, M. Danelutto, D. Laforenza, and M. Vanneschi, Eds., vol. 3149. Pisa, Italy: Springer, August/September 2004, pp. 230–237.

[5] T. N'Takpé, F. Suter, and H. Casanova, "A Comparison of Scheduling Approaches for Mixed-Parallel Applications on Heterogeneous Platforms," in *Proceedings of the 6th International Symposium on Parallel and Distributed Computing, Hagenberg, Austria*, 2007.

[6] S. Hunold, T. Rauber, and G. Rünger, "TGrid – Grid Runtime Support for Hierarchically Structured Task-parallel Programs," in *Proceedings of the Fifth International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (Heteropar'06)*. IEEE Computer Society Press, 2006.

[7] S. Hunold, T. Rauber, and G. Rünger, "Design and Evaluation of a Parallel Data Redistribution Component for TGrid," in *Proceedings of the International Symposium on Parallel and Distributed Processing and Applications (ISPA), Sorrento, Italy*, 2006.

[8] A. Legrand, L. Marchal, and H. Casanova, "Scheduling distributed applications: the simgrid simulation framework," *CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, vol. 00, p. 138, 2003.

[9] "Dag generation program." [Online]. Available: http://www.loria.fr/~suter/dags.html

[10] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1997.

[11] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, 2002.

[12] R. Prodan and T. Fahringer, "Dynamic Scheduling of Scientific Workflow Applications on the Grid: A Case Study," in *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*. New York, NY, USA: ACM Press, 2005, pp. 687–694.

[13] T. Rauber and G. Rünger, "Anticipated Distributed Task Scheduling for Grid Environments," in *Proc. of the IPDPS Workshop on High-Performance Grid Computing (HPGC)*. IEEE, 2006.