Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

Ocean-Atmosphere Modelization over the Grid

Yves Caniou ,
Eddy Caron ,
Ghislain Charrier ,
Andreea Chis ,
Frédéric Desprez ,
Éric Maisonnave

July 2008

Research Report N° RR2008-26

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr

Ocean-Atmosphere Modelization over the Grid

Yves Caniou , Eddy Caron , Ghislain Charrier , Andreea Chis , Frédéric Desprez , Éric Maisonnave

July 2008

Abstract

In this report, we tackle the problem of scheduling an Ocean-Atmosphere application used for climate prediction on the grid. An experiment is composed of several 1D-meshes of identical DAGs composed of parallel tasks. To obtain a good completion time, we divide groups of processors into sets each working on parallel tasks. The group sizes are chosen by computing the best makespan for several grouping possibilities. We improved this heuristic method by different means. The improvement yielding to the best makespan is the representation of the problem as an instance of the Knapsack problem. As this heuristic is firstly designed for homogeneous platforms, we present its adaptation to heterogeneous platforms. Simulations show improvements of the makespan up to 12%.

Keywords: Grid computing, Ocean-Atmosphere application, Scheduling

Résumé

Dans ce rapport, nous nous attaquons au problème d'ordonnancement d'une application Ocean-Atmosphere utilisée pour les prévisions d'évolution du climat sur la grille. Une xpérience est composée de plusieurs chaînes de DAGs identiques composés de tâches parallèles. Pour obtenir un bon temps d'exécution, nous divisons des groups de processeurs en ensembles, chacun travaillant sur une tâche parallèle. Les tailles des groupes sont choisis en calculant le meilleur temps d'exécution pour plusieurs possibilités de regroupement. Nous avons optimisé cette heuristique par différents moyens. L'optimisation amenant es meilleurs résultats et celle utilisant une représentation basée sur le problème du sac à dos. Cette heuristique a tout d'abord été conçue pour travailler sur des plate-formes homogènes, nous présentons aussi une adaptation pour plate-formes hétérogènes. Les simulations ont montrées des améliorations allant jusqu'à 12%.

Mots-clés: Calcul sur grille, application Ocean-Atmosphere, Ordonnancement

1 Introduction

World's climate is currently changing due to the increase of the greenhouse gases in the atmosphere. Climate fluctuations are forecasted for the years to come. For a proper study of the incoming changes, numerical simulations are needed, using general circulation models of a climate system (atmosphere, ocean, continental surfaces) on forced mode or coupled mode (*i.e.*, allowing information exchanges between each component during simulation).

Imperfection of the models and global insufficiency of observations make it difficult to tune model parametrization with precision. Uncertainty on climate response to greenhouse gases can be investigated by performing an ensemble prediction with varying parameters. Climatologists' strategy, in our case, is to launch parallel simulations. Each independent simulation models the evolution of the present climate followed by the 21^{st} century. All simulations have a distinct physical parametrization of clouds dynamics, which primacy in such studies has been emphasized in [4]. Comparing independent simulations, they expect to better understand the relations between the variation in this parametrization with the variation in climate sensitivity to greenhouse gases.

Our goal regarding the climate forecasting application is to thoroughly analyze it in order to model its needs in terms of execution model, data access pattern, and computing needs. Once a proper model of the application has been derived, appropriate scheduling heuristics can be proposed, tested, and compared.

The reminder of this paper is as follows. After a presentation of the target application in Section 2, we present related works in Section 3. In Section 4, we introduce the scheduling algorithm designed for the application. Then we present our work to execute the application on Grid'5000 in Section 5 before showing simulations results in Section 6. Finally, we conclude and discuss about further developments of this work.

2 Ocean-Atmosphere Simulations

The proposed climate modeling application consists of simulations of present climate followed by the 21^{st} century, for a total of 150 years (one scenario). A scenario combines 1800 simulations of one month each (150×12), launched one after the other. The results from the n^{th} monthly simulation are the starting point of the $(n+1)^{th}$. For the whole experiment, several scenarios are performed simultaneously.

A monthly simulation can be divided into a pre-processing phase, a main-processing parallel task, and a post-processing phase of analysis. Figure 1 shows the different tasks during the execution of a simulation and the data dependencies between two consecutive months. The number after the name of each task represents the duration of the task in seconds. The times have been obtained by performing benchmarks.

During the **pre-processing phase**, input files are updated and gathered in a single working directory by **concatenate_atmospheric_input_files** (caif) and the model parametrization is modified by **modify_parameters** (mp). The whole pre-processing phase only takes few seconds.

The main computing task **process_coupled_run** (pcr) performs a one month long integration of the climate model. This model is composed by an atmosphere (ARPEGE [3], model belonging to Meteo-France and derived from their weather forecast version), an ocean and its sea-ice (OPA/NEMO [6], developed by CNRS at LOCEAN laboratory and shared by several european climate models), and a river runoff model (TRIP [7]). The OASIS coupler [14] ensures simultaneous run of each element and synchronizes information exchanges.

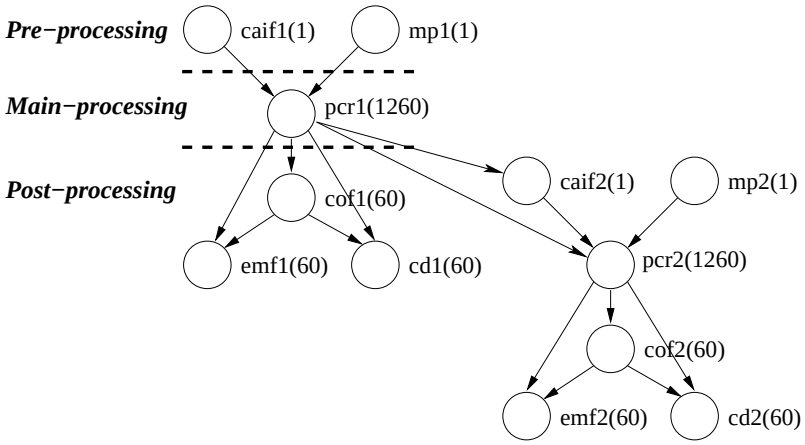

Figure 1: Chain of two consecutive monthly simulations.

ARPEGE code is fully parallel (using MPI communication library), while OPA, TRIP, and the OASIS coupler are sequential (in the chosen configuration of our climate model). The execution time of **process_coupled_run** depends on the number of processors allocated to the atmospheric model. We can note that with more than 8 processors, the speedup stops. OPA, TRIP and OASIS each need one processor, so pcr needs from 4 to 11 processors.

The **post-processing phase** consists of 3 tasks. First, a conversion phase **convert_output_format** (cof) where each diagnostic file coming from the different elements of the climate model is standardized in a self-describing format. Then, an analysis phase **extract_minimum_information** (emi) where global or regional means on key regions are processed. Finally, a compression phase **compress_diags** (cd) where the volume of model diagnostic files is drastically reduced to facilitate storage and transfers.

Data exchanges between two consecutive monthly simulations belonging to the same scenario reaches 120 MB. Simulations are independent, so there are no other data exchange.

3 Related Work

The execution of our application is represented by the execution of multiple DAGs containing tasks and data parallel tasks.

3.1 Multiple DAGs Scheduling

A Directed Acyclic Graph (DAG) is composed by nodes and edges, where each node represents a task and edges represent tasks dependancies. Scheduling several applications structured as DAGs can be solved in many ways [15].

A first approach is to schedule each DAG on the resources one after the other. The order in which DAGs are scheduled may influence the makespan. Another possibility is to concurrently schedule the DAGs. It is also possible to link all the entry tasks of the DAGs to an unique entry node and do the same with the exit nodes. Then, the new resulting DAG must be scheduled. To schedule the new DAG, it is possible to apply a Round-Robin policy among the tasks.

Since we want to have some fairness in the execution of the simulations of our application and DAGs are identical, we will use a variation of the latter.

3.2 Mixed Parallelism

Parallel scientific applications usually exhibit two types of parallelism: data parallelism and task parallelism. The first type occurs whenever the same operation is applied in parallel on different elements of a data-set while the second one appears in the form of concurrent computations running on different data sets. The combination of these two approaches is called mixed parallelism, which offers better speedups compared to the pure task parallelism or pure data parallelism.

Scheduling a DAG on a finite number of homogeneous resources is known to be NP-complete even for the simple case of tasks that execute only on a single processor. Heuristics have been developed to tackle this problem. For the case of DAGs composed of data parallel tasks and homogeneous platforms, several scheduling heuristics exist as well.

In [10], a two steps approach has been proposed. First, the number of processors on which a data-parallel task should be executed is computed, and then, a list scheduling heuristic is used to map the tasks onto processors. In [11], an approach of scheduling task graphs with a specific topology (series-parallel) is proposed. For series compositions, the tasks are allocated the whole set of processors, while for parallel compositions, the processors are partitioned into disjoint sets on which the tasks are scheduled.

In [8], a one step algorithm is proposed (Critical Path Reduction - CPR) for scheduling DAGs with data-parallel tasks onto homogeneous platforms. This algorithm allocates more and more resources to the tasks on the critical path and stops once the makespan is not improved anymore.

In [9], a two steps algorithm is proposed (Critical Path and Area based Scheduling - CPA). First the number of processors allocated to each data-parallel tasks is determined. In the second step, the tasks are scheduled on resources using a list scheduling heuristic.

These heuristics are not applicable here because our application does not contains a single critical path since all scenario simulations are independent. Each simulation is identical, so there are as many critical paths as simulations. In our case, we do not want to give processors to scenarios, but to create disjoint groups of resources on which scenarios will be executed. We choose this method because if there are 10 scenarios and 20 resources, it is faster to create 5 groups of 4 processors instead of 10 groups of 2 (the best grouping would be 2 groups of 10 resources).

3.3 Pipelined Data Parallel Tasks

Computations consisting of a chain of data-parallel tasks that process successive data sets in a pipeline fashion are a particular case where mixed parallelism occurs.

For this type of application communication cost is very important, thus we need to minimize it to improve the makespan.

In [13], the authors propose a dynamic programming solution for the problem of minimizing the latency with a throughput constraint and present a near optimal solution to the problem of maximizing the throughout with a latency constraint on homogeneous platforms.

Several aspects must be kept in mind when mapping the tasks of a pipeline on the resources. Subchains of consecutive tasks in the pipeline can be clustered into modules (which could thus reduce communications and improve latency) and the resources can be splitted among the resulting modules. Resources available to a module can be divided into several groups, on which processes will alternate data sets, improving the throughput but reducing the latency (which corresponds to the replication of the module).

Each simulation in Ocean-Atmosphere is a pipeline of data parallel tasks. We have to adapt the proposed heuristic to be able to work on several simulations at once.

4 Ocean-Atmosphere Application Scheduling

In this section we will present the heuristic we developed, three modifications of this method are developed and simulations done to validate the best version.

4.1 Scheduling Algorithm

We consider a homogeneous platform composed of R resources and that data on a site are available to all of its nodes. Thus, the execution time of any task is assumed to include the time to access the data, the time to redistribute it to processors (in case the task is a multiprocessor task), the computing time, and the time needed to store the resulting data. Given the short duration of the pre-processing tasks compared to the duration of the main-processing task, we made the decision to group them all in a single task. The same decision was taken for the 3 post-processing tasks. So, there are now 2 tasks: the main-processing task and the post-processing task. Figure 2 presents the new dependencies between tasks after the fusion.

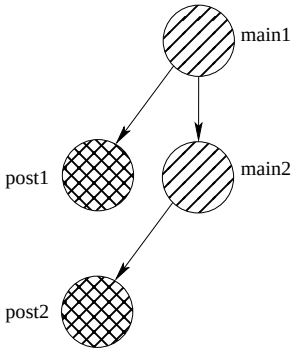

Figure 2: Two Consecutive simulations after grouping.

The purpose of this scheduling algorithm is to divide the resources of the platform into disjoint sets on which multiprocessor tasks will be executed such that the overall makespan would be minimal, under the assumption that all multiprocessor tasks will be executed on the same number of processors. The followings notations are introduced:

NS - number of independent simulations;

NM - number of months in an independent simulation;

R - total number of processors;

R_1 - number of processors (among the total R processors) allocated to the multiprocessor tasks;

R_2 - number of processors allocated to the post-processing tasks;

nb_{max} - maximum number of multiprocessor tasks that can run simultaneously given the current choice for the number of processors to be allocated to a multiprocessor task;

G - number of processors allocated to a single multiprocessor task;

T_G - execution time of a multiprocessor task on G processors;

T_P - execution time for a post-processing task;

Other notations are defined in order to simplify the lecture of the formulæ.

nb_{tasks} - number of each type of task ($nb_{tasks} = NS \times NM$);

nb_{used} - number of groups used on the last iteration of the main-processing tasks ($nb_{used} = nb_{tasks} \bmod nb_{max}$);

Since we can not process more than NS simulations simultaneously, we have nb_{max} = min $\{NS, \lfloor R/G \rfloor\}$. Resources allocated to multiprocessor tasks is then $R_1 = nb_{max} \times G$. The remaining resources are allocated to post-processing tasks, so we have $R_2 = R - R_1$.

There are 2 cases to be considered: $R_2 = 0$ and $R_2 \neq 0$ respectively.

Case 1. $R_2 = 0$**;**

In this case, multiprocessor tasks are executed first, followed by the post-processing tasks. The makespan of the multiprocessor tasks is given by:

$$MS_{multi} = \left\lceil \frac{nb_{tasks}}{nb_{max}} \right\rceil \times T_G; \tag{1}$$

If $nb_{used} = 0$, the total makespan is given by:

$$MS = \frac{nb_{tasks}}{nb_{max}} \times T_G + \left\lceil \frac{nb_{tasks}}{R} \right\rceil \times T_P; \tag{2}$$

Hatched rectangles in Figure 3 represent multiprocessor tasks and light empty rectangles represent the corresponding post-processing tasks:

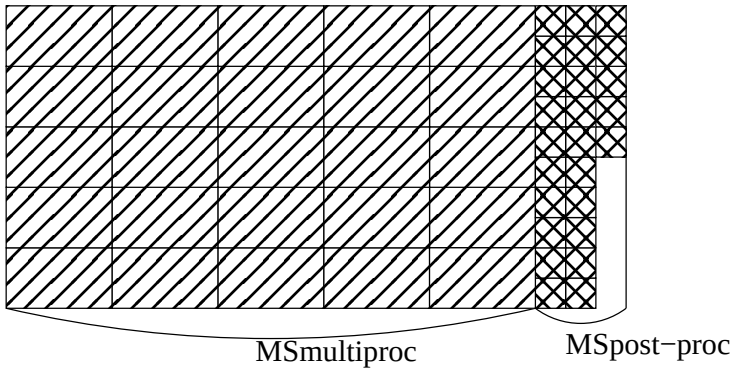

Figure 3: Makespan without processors allocated to the post-processing.

If $nb_{used} \neq 0$, a total of $remPost$ post-processing tasks do not fit on the resources left unoccupied on the last set of multiprocessor tasks ($R_{left} = R - nb_{used} \times G$). With the nb_{used} post-processing tasks corresponding to the last multiprocessor tasks, $remPost$ finally is: $remPost = nb_{used} + max\{0, nb_{tasks} - nb_{used} - \lfloor T_G/T_P \rfloor \times R_{left}\}$;

The makespan (MS) in this situation is:

$$MS = \frac{nb_{tasks}}{nb_{max}} \times T_G + \left\lceil \frac{remPost}{R} \right\rceil \times T_P; \tag{3}$$

Case 2. $R_2 \neq 0$**;**

In this case, the makespan of the multiprocessor tasks is the same as in Equation 1.

For a set of nb_{max} multiprocessor tasks, the execution time of the corresponding post-processing tasks is given by: $MS_{postproc\_phase} = \lceil nb_{max}/R_2 \rceil \times T_P$;

This time may be greater than the execution time of a multiprocessor task, in which case the execution time for the post-processing tasks will overpass the execution time of the next set of multiprocessor tasks (Figure 4).

The number of post-processing tasks that can be executed during the interval T_G on the R_2 resources reserved for them is: $N_{possible} = \lfloor T_G/T_P \rfloor \times R_2$;

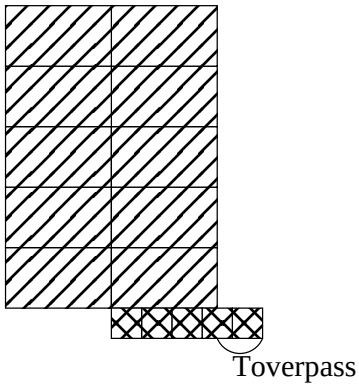

Figure 4: Post-processing tasks overpassing case.

This value must be tested against the nb_{max} value (since there are nb_{max} multiprocessor tasks generating the same number of post-processing tasks) in order to determine if the R_2 resources left are sufficient or not for the post-processing tasks. If they are not sufficient, the post-processing tasks which do not fit on the resources are reported for the end of the multiprocessor tasks. Otherwise, there may be a part of the R_2 resources which is not used during the whole process. This number of resources left idle is given by: $R_{unused} = R_2 - \left\lceil \frac{nb_{max}}{\lfloor T_G/T_P \rfloor} \right\rceil$;

We denote by n the total number of sets of simultaneous multiprocessor jobs: $n = \lceil nb_{tasks}/nb_{max} \rceil$;

Again, two separate cases must be treated, namely $nb_{used} = 0$ and $nb_{used} \neq 0$.

When $nb_{used} = 0$, the number of tasks reported for the end of the multiprocessor tasks (in the case such tasks exist) is: $N_{overpass} = max\{0, (n-1) \times (nb_{max} - N_{possible})\}$;

In this case, the total makespan is given by:

$$MS = MS_{multi} + \left\lceil \frac{N_{overpass} + nb_{max}}{R} \right\rceil \times T_P; \tag{4}$$

In the case $nb_{used} \neq 0$, a total of $N_{overpass}$ post-processing tasks corresponding to the first $n-2$ sets of simultaneous multiprocessor tasks will overpass the execution of the last $n-2$ complete sets of simultaneous tasks (Figure 5): $N_{overpas} = max\{0, (n-2) \times (nb_{max} - N_{possible})\}$;

Along with the nb_{max} post-processing tasks from the last complete set of simultaneous multiprocessor tasks, this gives a total of $N_{overtot} = N_{overpass} + nb_{max}$ tasks that should be scheduled starting on the resources left unoccupied in the last set of multiprocessor tasks ($R_{left} = R - G \times nb_{used}$) (Figure 6).

On one processor of the R_{left} remaining ones there can be scheduled $\lfloor T_G/T_P \rfloor$ post-processing tasks. The remaining tasks along with the post-processing task corresponding to the last (incomplete) set of multiprocessor tasks (nb_{used}) is: $remPost = nb_{used} + max\{0, N_{overtot} - (\lfloor T_G/T_P \rfloor \times R_{left})\}$;

Finally, the global makespan when $nb_{used} = 0$ is given by:

$$MS = MS_{multi} + \left\lceil \frac{remPost}{R} \right\rceil \times T_P; \tag{5}$$

All the 8 possibilities for the parameter G ($4 \rightarrow 11$) are tested and the one yielding the smallest makespan is chosen. The optimal grouping for various number of resources ($11 \rightarrow 120$) is plotted in Figure 7.

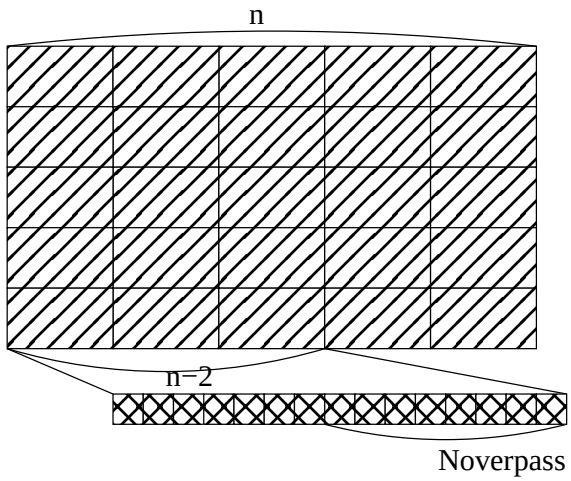

Figure 5: Post-processing tasks overpassing.

4.2 Heuristic Optimizations

For a given optimal grouping it may be possible that for a set of concurrent multiprocessor tasks and the associated post-processing tasks, all the available resources are not used. For example, for $R = 53$ resources, and 10 "scenario" simulations, the optimal grouping is $G = 7$. Hence a total of 7 multiprocessor tasks can execute concurrently, occupying 49 resources. The corresponding post-processing tasks need only 1 resource, which leaves 3 resources unoccupied during the whole simulation.

Improvement 1. In order to improve the makespan, the unoccupied resources can be distributed among the others groups of resources. Considering the previous example, we can redistribute the 3 resources left unoccupied among the 7 groups of resources for the multiprocessor tasks resulting in 3 groups with 8 resources and 4 groups with 7 resources and 1 resource for the post processing tasks giving a gain of 4.5% (58 hours less on the makespan).

Improvement 2. Given that the multiprocessor tasks scale well and that the post-processing tasks have a small duration, another possibility for reducing the makespan is to use the resources normally reserved for post-processing tasks for multiprocessor tasks and to leave all the post-processing at the end. It permits to avoid that the resource used to compute the post-processing become idle waiting for new tasks.

Improvement 3. The optimal repartition of the R processors in groups on which the multiprocessor tasks should be executed can be viewed as an instance of the Knapsack problem with an extra constraint. Given a set of items with a cost and a value it is required to determine the number of each item to include in a collection such that the cost is less than some given cost and the total value is as large as possible. Using a Knapsack representation of a problem as been studied in numerous areas such as scheduling [5] and aerodynamics [12].

In our case, there are 8 possible items (groups of 4 to 11 nodes). The cost of an item is represented by the number of resources of that grouping. The value of a specific grouping G is given by $1/T[G]$, which represents the fraction of a multiprocessor task that gets executed during a time unit for that specific group of processors. The total cost is represented by the total number of resources R.

The goal of the division of the processors in groups is to compute the biggest fraction of the multiprocessor tasks during a time interval.

We have n_i unknowns (i from 4 to 11) representing the number of groups with i resources which

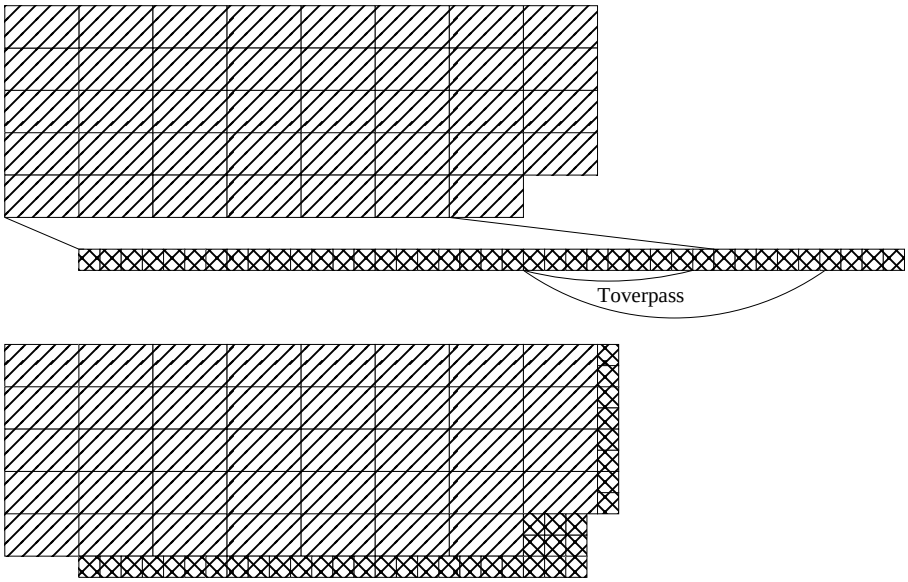

Figure 6: Post-processing tasks overpassing and final schedule.

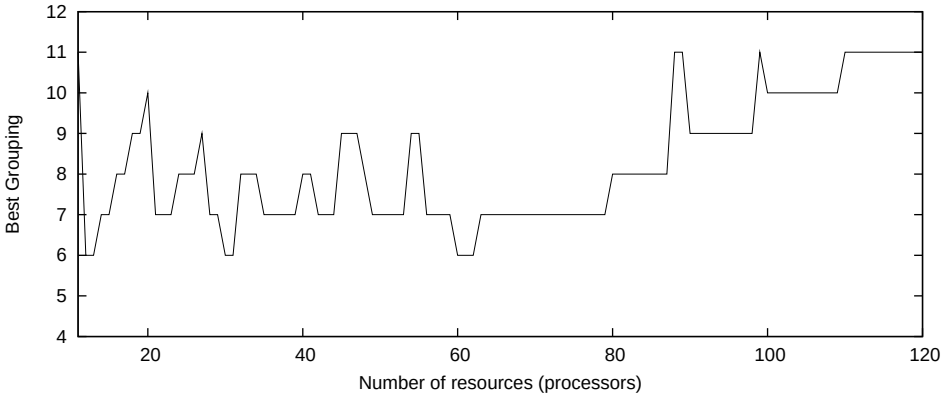

Figure 7: Optimal groupings for 10 scenario simulations.

will be taken in the final solution. The goal is to maximize $\sum_{i=4}^{11} n_i \times (1/T[i])$ under the constraints $\sum_{i=4}^{11} i \times n_i \leq R$ and $\sum_{i=4}^{11} n_i \leq NS$ (given that no more than NS tasks can be executed simultaneously).

4.3 Simulations

The execution of multiprocessor tasks is done by sorting the ready time of each group of processors and when a group becomes ready, the month of the less advanced simulation waiting is scheduled on this group.

Gains on the makespan obtained with the 3 possible improvements presented with respect to the first version of scheduling are plotted in Figure 8. These results come from 5 simulations done on clusters with different computing powers. The figure shows the average of the gains, and also the standard deviation.

The representation as an instance of the Knapsack problem yields to the bests results with low

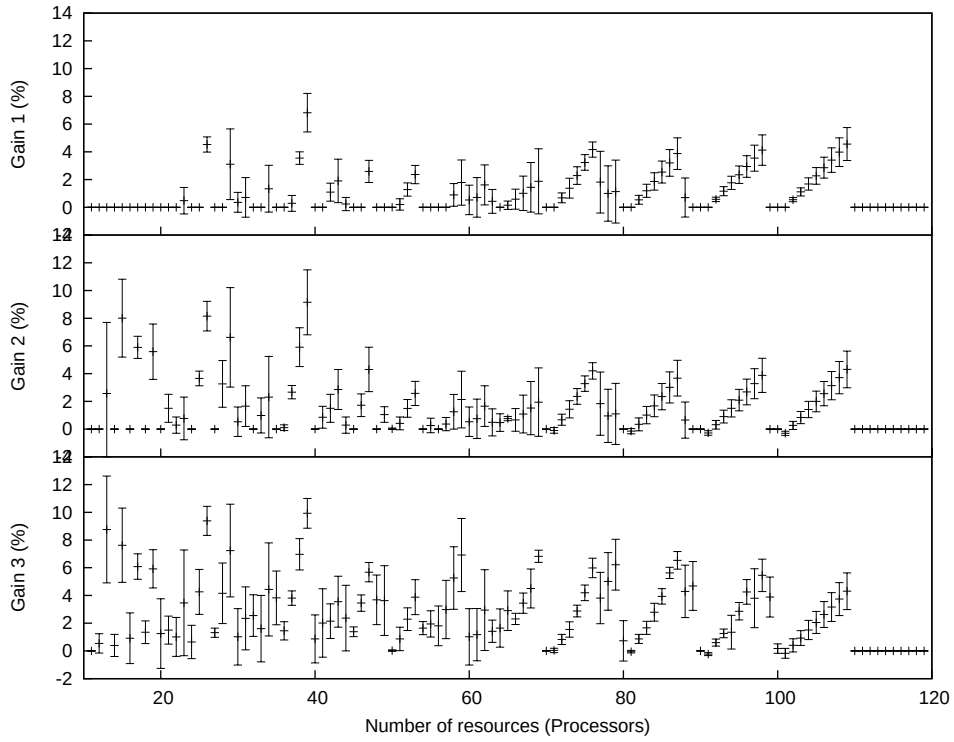

Figure 8: Gains obtained by using resources left unoccupied (Gain 1), using all resources for post-processing tasks (Gain 2), and using the Knapsack problem (Gain 3).

resources. The gains are less important with more resources, and it even becomes a little less good with a lot of resources. With a lot of resources, there are no more gains since there are NS groups of 11 resources.

5 Application on the Grid

The scheduling presented in Section 4 is designed for homogeneous platforms. Grid'5000 [1] is a grid composed of several clusters. Each cluster is composed of homogeneous resources but differs from one another. We intend to deploy Ocean-Atmosphere on Grid'5000 so we have to adapt the algorithm to be able to work on heterogeneous platforms. Implementation of the scheduling heuristics will be done in the DIET grid middleware [2] to perform real experiments.

To reduce the makespan of NS simulations, the best way is to divide the set of simulations into subsets and execute each subset on a different cluster. The choice of the number of simulations executed on each cluster is given by Algorithm 1.

The different steps of the execution are displayed in Figure 9. (1) the client sends a request to the clusters with the number of simulations NS and the number of months for each simulation NM; (2) each cluster computes a vector containing the time needed to execute from 1 to NS simulations using the Knapsack modeling given before (3) returning the results to the client; (4) Once the client received the results from all clusters, it computes the repartition of the simulations; (5) Once the repartition done, the client sends the requests to each cluster to execute the simulations; (6) Finally, each cluster computes the simulations it has been assigned.

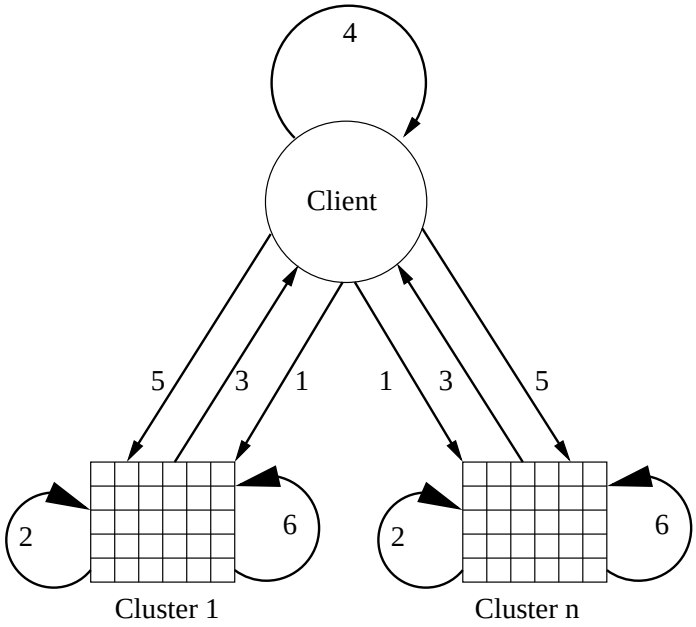

Figure 9: Execution Steps for the application.

Algorithm 1 describes the way the repartition is done between clusters. Input parameters are: n, the number of clusters, and the array "performance" which has been initialized for each cluster with the makespan for the execution of 1 to NS simulations. First, the number of simulations on each cluster is set to 0. Then, each simulation is scheduled on the cluster on which the total makespan increases the less. When all the simulations are scheduled, this scheduling is returned to the client.

This algorithm is realistic because the number of simulations (NS) and clusters (n) are quite low. The number of clusters on Grid'5000 is low, and the number of simulations is going to be around 10. The algorithm gives the optimal repartition for the times given in the "performance" array. If we map a scenario onto another cluster, the total makespan cannot decrease. This is true only if we consider that once a scenario has been scheduled on a cluster, it can not change location.

Algorithm 1 DAGs repartition on several clusters.

for $i = 1$ to n **do**
  $nbDags[i] = 0$
for $dag = 1$ to NS **do**
  $MSmin = +\infty$
  $clusterMin = 1$
  for $i = 1$ to n **do**
    $temp = performance[i][nbDags[i] + 1]$
    if $temp < MSmin$ **then**
      $MSmin = temp$
      $clusterMin = i$
  $nbDags[clusterMin] = nbDags[clusterMin] + 1$
  $repartition[dag] = clusterMin$
Return $repartition$

6 Simulations for the grid

The repartition of the DAGs among the different clusters is done using Algorithm 1. In order to be as accurate as possible, we benchmarked the execution time of the application on numerous clusters of Grid'5000. These clusters are located all around France and the difference of performances is notable, *e.g.*, the fastest cluster executes one main-processing task on 11 resources in 1177 seconds while the slowest needs 1622 seconds.

To conduct simulations, we used performance of five clusters. Figure 10 shows the gains obtained by the different heuristics presented in Section 4.2 compared to the basic heuristic presented in Section 4.1. Clusters have all the same number of resources. The X axis represents the number of clusters and the number of resources per cluster, hence 2.25 represents the results for two clusters with 25 resources each.

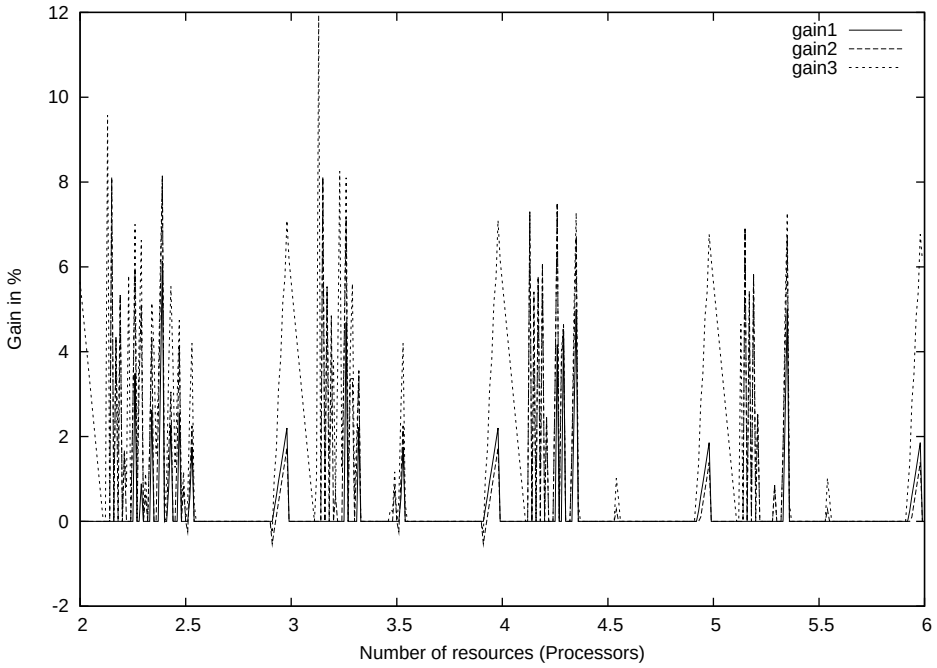

Figure 10: Gains obtained using DAGs repartition on 2 to 5 clusters each with 11 to 99 resources.

The best gains obtained are almost of 12%, but the most common gains are from no gain to 8%. Starting from two clusters, there are stable phases where no heuristic improves the basic one. This is the case when the makespan depends on the slowest cluster, and the heuristics make the same grouping of resources on this cluster. Just after this phase, the improved heuristics make other grouping on a faster cluster which permits to take one simulation from the slow cluster to a faster one.

These simulations show that if clusters are added, the gains obtained by the different heuristics are less and less important. The gains decrease because there are more resources, so the basic heuristic behaves better.

7 Conclusion

World's climate is currently changing. In order to predict its variations, simulations are conducted by climatologists and the computation of such simulations is very time consuming.

This paper presents the work of analyzing and modeling a real climatology application (Ocean-Atmosphere) with the purpose of deriving appropriate scheduling heuristics in order to decrease the execution time of such applications.

First, the computation needs have been modeled as independent identical workflows derived through the chaining of several basic DAGs. Then a simplified model with clustered tasks based upon the actual time parameters of the application has been derived.

For this new model, a first scheduling heuristic (driven by the principle of allocating the same number of processors to all multiprocessor tasks and leaving what is left to post-processing tasks) has been designed. Three improved versions have been proposed. A first one that distributes resources left unused evenly across the groups of processors, a second one which does not leave any resource for the post processing tasks and distributes all left resources evenly to the groups of processors and a third one that models the problem of dividing the resources of the platform in disjoint sets as an instance of the Knapsack problem with a supplementary constraint. The three improved versions have been simulated and yielded to gains of up to 12%.

Then, this method was adapted to be applicable on a heterogeneous grid composed of homogeneous clusters. Distributing the simulations among different clusters reduces the overall makespan of the application.

Simulations for the grid have been conducted. They show that the distribution of the simulations is function of the clusters performance. The faster, the more DAGs it has to execute.

The integration of the scheduling heuristics within DIET is ongoing work. Once the development completely done, we will be able to verify our simulations by real experiments on Grid'5000. Future work also consists in extending the present work to a generic heuristic that can schedule the same kind of workflow, made of independent chains of identical DAGs composed of moldable tasks.

References

[1] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, El-G. Talbi, and Touché I. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, November 2006.

[2] E. Caron and F. Desprez. Diet: A scalable toolbox to build network enabled servers on the grid. *International Journal of High Performance Computing Applications*, 20(3):335–352, 2006.

[3] M. Deque, C. Dreveton, A. Braun, and D. Cariolle. The ARPEGE/IFS atmosphere model: a contribution to the french community climate modeling. *Clim Dyn*, 10:249–266, 1994.

[4] C.G. Knight, S.H.E. Knight, N. Massey, T. Aina, C. Christensen, D.J. Frame, J.A. Kettleborough, A. Martin, S. Pascoe, B. Sanderson, D.A. Stainforth, and M.R. Allen. Association of parameter, software and hardware variation with large scale behavior across 57,000 climate models. *Proceedings of the National Academy of Sciences*, 104:12259–12264, 2007.

[5] C. Lu and S.M. Lau. An Adaptive Load Balancing Algorithm for Heterogeneous Distributed Systems with Multiple Task Classes. *International Conference on Distributed Computing Systems*, pages 629–636, 1996.

[6] G. Madec. *NEMO Reference manual, ocean dynamics component: NEMO-OPA*. Number 27. Institut Pierre Simon Laplace (IPSL), 2006. ISSN 1288-1619.

[7] T. Oki and Y. C. Sud. Design of Total Runoff Integrating Pathways (TRIP)-A Global River Channel Network. *Earth Integration*, 2(1234):1–37, 1998.

[8] A. Radulescu, C. Nicolescu, A. J. C. van Gemund, and P. Jonker. CPR: Mixed task and data parallel scheduling for distributed systems. In *IEEE International Parallel and Distributed Processing Symposium*, page 39. IEEE Computer Society, 2001.

[9] A. Radulescu and A. J. C. van Gemund. Low-cost mixed task and data parallel scheduling. In *30-th International Conference on Parallel Processing (ICPP)*, pages 69–76, August 2001.

[10] S. Ramaswamy, S. Sapatnekar, and P. Banerjee. A Framework for Exploiting Data and Functional Parallelism on Distributed Memory Multicomputers. *Urbana*, 51:61801, 1994.

[11] T. Rauber and G. Runger. Compiler support for task scheduling in hierarchical execution models. *Journal of Systems Architecture*, 45(6-7):483–503, 1999.

[12] J.M. Romaine. *Solving the Multidimensional Multiple Knapsack Problem with Packing constraints using Tabu Search*. PhD thesis, 1999.

[13] J. Subhlok and G. Vondran. Optimal use of mixed task and data parallelism for pipelined computations. *Journal of Parallel and Distributed Computing*, 60(3):297–319, 2000.

[14] S. Valcke, R. Caubel, A. Vogelsang, and D. Declat. Oasis 3, user guide. Technical Report PRISM Report Serie no 2 (5th edition), CERFACS, Toulouse, 2004.

[15] H. Zhao and R. Sakellariou. Scheduling multiple DAGs onto heterogeneous systems. In *IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2006.