On the Distribution of Dependent Tasks Over Non-dedicated Grids with High Bandwidth Links

Luiz F. Bittencourt and Edmundo R. M. Madeira
Institute of Computing
State University of Campinas - UNICAMP
P.O. 6176, Campinas - São Paulo - Brazil
Email: {bit,edmundo}@ic.unicamp.br

***Abstract*— The distribution of workflow-like dependent tasks over a set of heterogeneous resources is an NP-Complete problem. Thus, many solutions are heuristic-based, leading to simulations for the validation and comparison of dependent task scheduling algorithms. However, simulations concerning the algorithms' performance when executed in non-dedicated resources with a wide range of bandwidth capacities are scarce. On this paper we evaluate the performance evolution of the dynamic *Path Clustering Heuristic* against the static one when scheduling dependent tasks on resources connected by gigabit networks. The experimental results indicate that the dynamic PCH mantains a constant performance gain over the static PCH with the change from a megabit to a gigabit network.**

I. Introduction

A computational grid is a heterogeneous, geographically distributed and dynamic system. Thus, the scheduler must consider the heterogeneity and communication delays when scheduling dependent tasks on a grid. The scheduler is the entity that handles the application and assigns it to a resource. The main goal of a scheduling system is to minimize the execution time of an application (*makespan*). In the task scheduling problem the scheduler handles an application composed of tasks with precedence constraints, trying to minimize the application's execution time (*makespan*) by distributing its components to the available resources. In a grid, these resources are heterogeneous, as the communication capacities between them. Thus, a task scheduler must deal with task precedence, communication and computation costs. This is an NP-Complete problem [1], so there is no known polynomial time solution. Also, a grid task scheduler must deal with an eventual performance loss in the resources, being adaptable to the dynamic computing power available.

The NP-Completeness of the task scheduling problem led us to the development of a heuristic-based algorithm for the associated optimization problem [2]. This algorithm, called *Path Clustering Heuristic* (PCH), was developed to work in a grid based on Xavantes [3], a middleware that supports dependent task execution through hierarchical control structures called *controllers*.

A grid with high bandwidth links is a powerful environment for executing processes with dependent tasks, since the high communication capacities minimizes the delay between the finish of a task and the start of its dependent task on another resource. With this, the data dependencies have less impact on the final makespan when compared to environments with lower bandwidth links.

A DAG meta-scheduler for grids is proposed in DAGMan [4]. It just sends one task at a time to the scheduler, which schedules the tasks like independent tasks, without knowledge about dependencies. In [5], a case study on dynamic scheduling for scientific workflow applications on the grid is presented. It proposes a static reschedule with iterations over the workflow application, generating a static DAG with the tasks that would be rescheduled on each iteration. After that, the generated DAG is scheduled, actually performing a reschedule of its tasks. In this work we make a case study of PCH, comparing the dynamic and static approaches on grids with high bandwidth links. The experiments were made considering a wide range of link capacities.

The paper is organized as follows. In Section II we give a brief introduction to Xavantes. Section III shows the PCH algorithm and its dynamic approach, while Section IV shows the experimental results. Section V concludes the paper.

II. Xavantes Middleware

Xavantes [3] is a grid middleware developed to support the execution of workflow-like processes composed of dependent tasks. It is composed of a programming model and an infrastructure, where the resources are organized as autonomous groups. For example, a laboratory, a LAN or a cluster could be an autonomous group. The processes are specified in a hierarchical way, similar to the structured programming languages, using the programming model defined in [3]. In the process specification, the programmer should specify the tasks and also some entities called *controllers*. The controllers are responsible for managing the execution of a process, controlling the execution state and the communication between tasks. For more details on Xavantes, please reffer to [3].

A process has potentially many controllers, some parallel and some sequential. Tasks inside a parallel controler have no dependency and could be executed in paralell, while tasks inside a sequential controller should be executed in sequence. When a task finishes its execution, it sends the results of its computation to the its controller, then the controller will check the workflow to determine to which task this data should be sent. The controllers are distributed among the available resources, distributing also the knowledge of the execution

state of the process. Thus, controllers provide an easy recovery mechanism to the workflow execution. Also, controllers have a shared memory space that could be used to allocate shared variables, giving the possibility of communication between parallel tasks even if they do not have a data dependency.

A process is represented by a directed acyclic graph (DAG), with the nodes being the tasks and the edges being the dependencies between tasks. The labels in the nodes are the computation costs and the labels in the edges are communication costs. The computation and communication costs are specified in the programming model. Additionally, the rectangles represent the controllers. In the DAG of Figure 1, the rectangle 1 represents a parallel controller containing the tasks 7 and 8, and the rectangle 4 represents a sequential controller containing the activities 2, 5 and 10, and the controller 1.

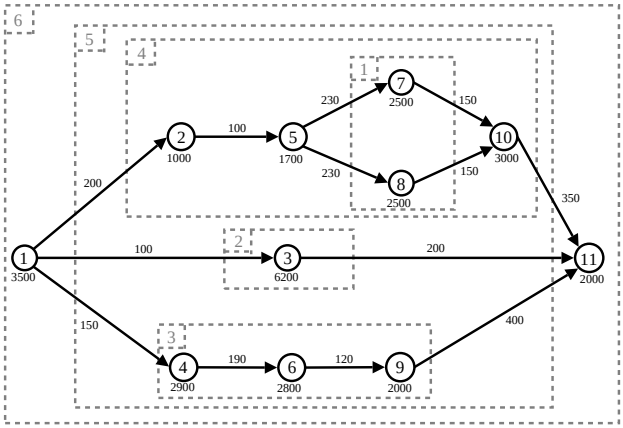

Fig. 1. Task graph example.

III. PCH Algorithm

The Path Clustering Heuristic (PCH) [2] algorithm was developed to work with Xavantes, being able to provide a good performance with easy recovery, taking advantage of the controllers. Since every communication between tasks must be via controllers, the presence of them can generate some communication overhead. For example, if two tasks that have a dependency are executing on the same resource, but their controllers are executing on another resource, a communication would be necessary even with tasks on the same resource. The data should go to the controllers on another resource and return after the controller's action.

Based on the above issues, the PCH tries to group dependent tasks, minimizing their communication and being able to execute more controllers on the same resource as their tasks. With this, the communication overhead of the controllers could be also minimized. The groups of tasks are called *clusters*, and are composed of tasks on the same path of the DAG, traversing the graph in a depth-fist manner. To choose what path to follow when there is a fork on the graph, the algorithm calculates some graph attributes. The decision of which tasks will compose a cluster is based on these attributes. An overview of PCH is given in Algorithm 1.

Algorithm 1 PCH Algorithm

```
1: Compute all tasks attributes
2: while there are unscheduled nodes do
3:     cluster ⇐ get_next_cluster()
4:     resource ⇐ get_best_resource(cluster)
5:     Schedule cluster on resource
6:     Recalculate tasks attributes
7: schedule_controllers()
```

The steps of the algorithm are as follows: first, the algorithm computes the attributes that will guide the creation of clusters of tasks. Then, the algorithm creates the cluster and selects a resource to it. The cluster is assigned to the resource chosen and the attributes are recalculated. This is repeated while there are unscheduled nodes. With all nodes scheduled, the algorithm must choose where the tasks' controllers will execute, selecting the resource where there are the most tasks of the controller that is being scheduled. This is the last step of the algorithm. For details about how the attributes are calculated, how the clusters are created or how resources are selected, please reffer to [2].

A. Dynamic PCH

Since we consider a non-dedicated grid, a static scheduling approach as presented in Section III, where all tasks are scheduled before the process execution's start, could not offer good makespans when there are performance losses on resources. Therefore, a dynamic approach could improve the system performance regarding the grid processes' makespans.

The static PCH gives good results in a static heterogeneous environment with a mesh heterogeneous (no-group) topology. In [2] we propose a dynamic algorithm to schedule dependent tasks on a grid with variations on resources' performance. The algorithm uses the static PCH to schedule tasks and introduces a round-based approach to minimize performance losses on resources. The PCH makes the initial schedule and then a *round-based* approach is applied to reschedule tasks when necessary. Algorithm 2 is an overview of this approach.

Algorithm 2 Dynamic Approach Overview

```
1: Schedule DAG G using the static PCH Algorithm
2: while not(all nodes of G have finished) do
3:     Select tasks to execute according to a policy.
4:     Send tasks of this round to execution.
5:     Evaluate the resources performance.
6:     Reschedule tasks if necessary.
```

To make a dynamic schedule of tasks we developed the concept of *rounds*. In each round some tasks are selected based on a criterion and sent to execution. Then, the scheduler verifies the performance obtained on each resource used on that round. If the performance of a resource is below a threshold, the algorithm reschedules the non executed tasks. Note that there is no *reallocation* of tasks, so there is no

overhead of moving tasks, since the reschedule is made before the tasks are sent to execution.

In Xavantes, the scheduler is responsible for detecting and rescheduling tasks that are in resources with low performance. Fails in resources are detected by the middleware and the tasks are sent to the scheduler, so it can reschedule them. The process of recovery is handled by the middleware, using the information provided by the controllers.

IV. Experimental Results

In this work we evaluated the performance of the dynamic PCH against the static PCH in a wide range of link bandwidths. The purpose of such an evaluation is to experimentally check the evolution of the dynamic PCH when the links capacities change from Mbps to Gbps. Fifteen DAGs composed of tasks with random computation and communication costs were taken for the experiment. The interval of computation costs were $(10.000, 310.000)$ millions of instructions. The communication costs were high, between 250 and 750 megabytes.

The grid for the experiment was composed of groups of resources with varible link loads and variable computing power. We considered 5 groups with at most 7 resources each. A group of resources had internal connection between 100% and 40% of the maximum link bandwidth considered, simulating a non dedicated network. The connection among groups was considered to have between 85% and 20% of the maximum link bandwidth. The computing power of each machine was a random value between 4.000 and 10.000 MIPS. Additionally, the resources capacities vary over the time in a random way, simulating the owner of the resource using its machine to execute his/her own processes. For each pair (bandwidth,rounds) each graph was scheduled 1.000 times.

The Figure 2 shows the average speedup for executions with 3, 5 and 7 rounds. The speedup shows how much times the exectution was faster than if the whole process was executed on the best resource available. The maximum bandwidth varies from 100 megabits per second to 50 gigabits per second.

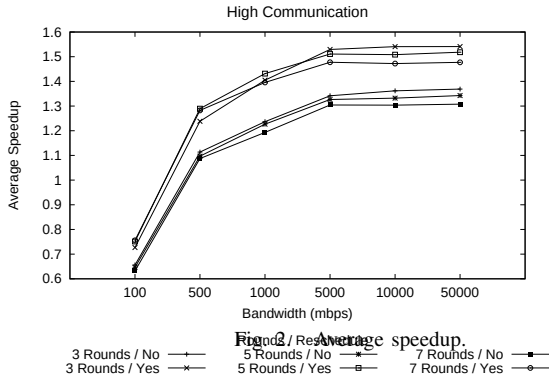

Fig. 2. Average speedup.

The Figure 3 shows the average SLR (schedule length ratio) for executions with 3, 5 and 7 rounds. The SLR shows how much times the execution was slower than the execution of the graph's critical path on the best resource available.

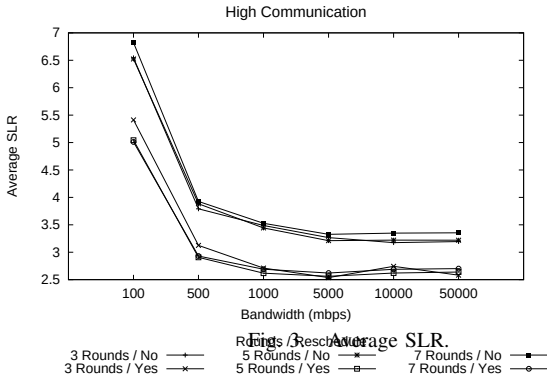

Fig. 3. Average SLR.

The results indicates that the dynamic approach of PCH mantains a good performance with the increase of links capacities. The speedup results show a wider difference with 50 Gbps than with 100 Mbps, although proportionally the difference is the same. The SLR results show that the schedule length ratio difference is smaller in the 50 Gbps scenario than in the 100 Mbps scenario. Again, proportionally this difference is the same. Also, in this scenario, there is a stabilization point near 10 Gbps, where the communication costs are much lower than the computation costs because the bandwidth is very high.

V. Conclusion

In this paper we make an experimental evaluation of dynamic and static versions of PCH algorithm in scenarios with different bandwidth capacities, ranging from 100 Mbps to 50 Gbps. The results show that the round-based dynamic PCH is not affected by high bandwidth links, mantaining a good performance when compared to its static version.

Future works include searching for an ideal number of rounds for each graph, depending on its properties. Also, a history of resources performance could be added to the schedule step, trying to do a performance prediction.

Acknowledgment

The authors would like to thank CAPES and FAPESP (Process 2003/08277-0) for the financial support.

References

[1] H. El-Rewini, H. H. Ali, and T. G. Lewis, "Task scheduling in multiprocessing systems." *IEEE Computer*, vol. 28, no. 12, pp. 27–37, 1995.

[2] L. F. Bittencourt and E. R. M. Madeira, "A dynamic approach for scheduling dependent tasks on the xavantes grid middleware," in *4th ACM International Workshop on Middleware for Grid Computing*, Melbourne, Australia. nov/dec 2006. (Accepted for publication).

[3] F. R. L. Cicerre, E. R. M. Madeira, and L. E. Buzato, "A hierarchical process execution support for grid computing," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 6, pp. 581–594, 2006.

[4] J. Frey, "Condor DAGMan: Handling inter-job dependencies. http://www.cs.wisc.edu/condor/dagman/," 2002.

[5] 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.