Experimental Study of Multi-Criteria Scheduling Heuristics for GridRPC Systems

Yves Caniou* and Emmanuel Jeannot

[1] LORIA, INRIA-Lorraine
[2] LORIA, Université Henri Poincaré

Abstract. We study in this paper several scheduling heuristics for GridRPC middlewares. When dealing with performance issue, the scheduling strategy is one of the most important feature. However, many heuristics implemented in available middlewares were not designed for this context (for instance MCT in NetSolve). Therefore, schedulers are not necessarily effective. We propose to use heuristics based on a non-intrusive module which is able to estimate the duration of all tasks in the system. Three criteria are examined among which the execution time of the application, e.g. the makespan. Experiments on a real platform show that the proposed heuristics outperform MCT for at least two of these three criteria.

Keywords : time-shared and heterogeneous resources, dynamic scheduling heuristics, historical trace manager, completion dates estimations, DAGs

1 Introduction

GridRPC [1] is an emerging standard promoted by the global grid forum (GGF).This standard defines both an API and an architecture. A GridRPC architecture is heterogeneous and composed of three parts: a set of clients, a set of servers and an agent (also called a registry). The agent has in charge to map a client request to a server. In order for a GridRPC system to be efficient, the mapping function must choose a server that fulfills several criteria. First, the total execution time of the client application, e.g. the makespan, has to be as short as possible. Second, each request of every clients must be served as fast as possible. Finally, the resource utilization must be optimized.

Several middlewares instantiate the GridRPC model (NetSolve [2], Ninf [3], DIET [4], etc.). In these systems, a server executes each request as soon as it has been received: it never delays the start of the execution. In this case, we say that the execution is *time-shared* (in opposition to *space-shared* when a server executes at most one task at a given moment). In NetSolve, the scheduling module uses MCT (Minimum Completion Time) [5] to schedule requests on the servers. MCT was designed for scheduling an application for space-shared servers. The goal was to minimize the makespan of a set of independent tasks. This leads to the following drawbacks:

* This work is partially supported by the Région Lorraine, the French ministry of research ACI GRID

- *mono-criterion and mono-client*. MCT was designed to minimize the makespan of an application. It is not able to give a schedule that optimizes other criteria such as the response time of each request. Furthermore, optimizing the last task completion date does not lead to minimize each client application makespan. However, in the context of GridRPC, the agent has to schedule requests from more than one client.
- *load balancing*. MCT tries to minimize the execution time of the last request. This leads to over-use the fastest servers. In a time-shared environments, this implies to delay previously mapped tasks and therefore degrades the response time of the corresponding requests.

Furthermore, MCT requires sensors that give information on the system state. It is mandatory to know the network and servers state in order to take good scheduling decisions. However, supervising the environment is intrusive and disturbs it. Moreover, the information are sent back from time to time to the agent: they can be out of date when the scheduling decision is taken.

In order to tackle these drawbacks we propose and study three scheduling heuristics designed for GridRPC systems. Our approach is based on a prediction module embedded in the agent. This module is called the Historical Trace Manager (HTM) and it records all scheduling decisions. It is not intrusive and since it runs on the agent there is no delay between the determination of the state and its availability. The HTM takes into account that servers run under the time-shared model and is able to predict the duration of a given task on a given server as well as its impact on already mapped tasks. In this paper, we show that the HTM is able to predict precise task durations when the server load is not too high. The proposed heuristics use the HTM to schedule the tasks. We have plugged the HTM and our heuristics in the NetSolve system and performed intensive series of tests on a real distributed platform (more than 50 days of continuous computations) for various experiments with several clients.

In [5], MCT was only studied in the context of independent tasks submission. In this paper, the study is more general. The tests are submissions of task graphs (1D meshes and stencil). For some of them, independent tasks are submitted during the execution of the graphs. We do not assume any knowledge of the applications graphs and thus, we schedule dynamically each request. The goal is to serve at best each client request (a task of a graph as well as an independent one). We have compared our heuristics against MCT in NetSolve on three criteria. Results show that the proposed heuristics outperform MCT on at least two of the three criteria with gain up to 20% for the makespan and 60% for the average response time.

2 MCT in GridRPC Systems

GridRPC is an emerging standard for Grid Computing. It allows the remote execution of tasks to computational servers. The architecture is composed of clients that submit requests to an agent (or a registry), the agent sends back the identity of the servers that can execute the tasks. The client sends input data to the chosen server which performs the computation and sends back output data.

NetSolve [2] instantiates this model. A NetSolve agent uses Minimum Completion Time [5] (MCT) to map tasks on servers. The MCT heuristic chooses the server that will complete the task the soonest. In order to determine the completion time of a given task on a server it assumes that the load on the network and on the server is constant during the execution. This leads to the following remarks:

1. the agent needs an evaluation of each server load ;
2. the load on a given server is seldom constant. If the server is loaded by some tasks, they are likely to finish during the computation of the new mapped task ;
3. The time-shared model implies that mapping a task on a loaded server delays the concurrent running tasks. In this paper, we call this delay the *perturbation*. The perturbation may make the previously decision obsolete.

The load evaluation of a network or of a machine is a difficult task and is often performed by sensors. NetSolve can use its own sensors or NWS [6]. It faces two major problems: accuracy (MCT considers that the server load is constant when mapping a task and do not consider the perturbation a new task will produce on the server) and intrusiveness (sensors require some cpu cycles on the servers to evaluate the load).

3 Historical Trace Manager

The Historical Trace Manager is an attempt to efficiently answer the three remarks exposed in the previous section. It is a prediction module that runs on the agent. It is accurate and non intrusive. It simulates the execution of the tasks on each server and therefore is able to predict load variation and to compute the perturbation of the new task to already mapped ones.

We use the following model to simulate time-shared resources. When n tasks are using the same resource (CPU, network, etc.), each one uses $1/n$ of its peak power. The HTM needs several types of information: server and network peak performances, the size of input and output data and the number of operations of each task. All these information are static and can be computed off-line Therefore, the HTM answers the three remarks exposed in Section 2:

1. the HTM is not intrusive because it uses only static information. Furthermore, since it runs on the agent, information are immediately available for the scheduling heuristic. The accuracy of the information given by the HTM is high and will be experimentally demonstrated in the results section ;
2. the HTM is able to compute the completion date of every tasks on a given server by simulating their executions. Therefore, the load is not assumed constant ;
3. The HTM can simulate the mapping of a task on any server. Hence, it can compute the perturbation of this task on all the concurrent running ones. The perturbation can then be used to take efficient scheduling decisions, as we will see in the next sections.

4 Heuristics

We introduce here three heuristics, HMCT, MP and MSF, which are compared to MCT in section 5. The HTM simulates the new task on each server and gives resultant information to the scheduler. Therefore, the heuristic considers the perturbation the new task will induce on each running one, and computes the 'best' server accordingly.

Historical MCT: HMCT. HMCT chooses the server that will complete the request the fastest. Unlike MCT which assumes a constant behavior of the environment during the execution of the task to predict its finishing date (see Section 2), HMCT uses HTM estimations which are far more precise.

Minimum Perturbation: MP. The HTM simulates the new task on each server. MP chooses the server that minimizes the sum of the HTM estimated delays that running tasks will suffer. It aims to minimize overall running tasks completion dates and each client can thus expect a lower *response time*.

Minimum Sum Flow: MSF. MSF relies on HTM estimations to compute the sum of the duration (also called the sumflow [7]) of all the tasks. It then chooses the server which minimizes the sumflow. MSF aims at reducing the duration of all the tasks (not only the new one).

Table 1. Resources Composing the Environment and task durations on the unloaded servers

Type	Machine	Processor	Speed	Memory	Swap	System	task 1	task 2	task 3
Server	spinnaker	xeon	2 GHz	1 Go	2 Go	Linux	15	30	43
	artimon	pentium IV	1.7 GHz	512 Mo	1024 Mo	Linux	17	33.5	49.5
	soyotte	sparc Ultra-1		64 Mo	188 Mo	SunOS	128	256	382.5
	fonck	sparc Ultra-1		64 Mo	188 Mo	SunOS	127.5	254	380.9

5 Experiments and Results

5.1 Experiments

The HTM and the heuristics HMCT, MP and MSF have been implemented in the NetSolve agent. Several experiments have been conducted onto an heterogeneous environment whose resources are given in Table 1. Three types of computing intensive tasks have been used for these experiments. The duration of each task on each unloaded server is given in Table 1. computing Table 2 gives a summary of the series of tests performed on this platform. 8 scenarii (from (a) to (h)) have been submitted to this heterogeneous environment. Two types of graphs have been used. 1D meshes and stencil graphs. In a scenario, only one client submits a stencil graph while there can be several clients submitting a 1D mesh application (5 or 10). Stencil graphs have a width of 5 or 10 and a depth of 25 or 50. In scenarii (c), (f), (g) and (h), independent tasks are submitted during the execution of the graph(s). Their arrival dates are drawn from a Poisson distribution of parameter μ. We use the GSL library [9] for all the probabilistic calls. Experiments in a scenario are randomized by a different seed and run 6 times per heuristics. 6 runs were sufficient for the average values given in Section 5.3 to be stabilized. This leads to 576 experiments and about 50 days of computations.

Table 2. Scenarii, Modalities and Number of Experiments

Scenario	Application(s)		Independent Tasks		Experiment	
	nbclients	width x depth	nbtasks	μ (sec)	nbseeds x nbrun	total nbtasks
(a) 1D-mesh	10	1x50	-	-	4 x 6	500
(b) 1D-mesh	10	1x variable	-	-	4 x 6	500
(c) 1D-mesh + 250 independent tasks	5	1x50	250	20	4 x 6	500
(d) stencil (task 1)	1	10x50	-	-	1 x 6	500
(e) stencil (task 3)	1	10x50	-	-	1 x 6	500
(f) stencil + 175 independent tasks	1	10x25	175	28	2 x 6	425
(g) stencil + 87 independent tasks	1	10x25	87	40	4 x 6	337
(h) stencil + 87 independent tasks	1	5x25	87	25	4 x 6	212

5.2 Accuracy of HTM Predictions

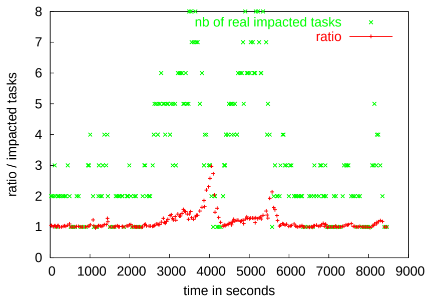

Fig. 1. 500 Independent Tasks Submitted at a Rate $\mu = 17$ Scheduled by MSF on artimon

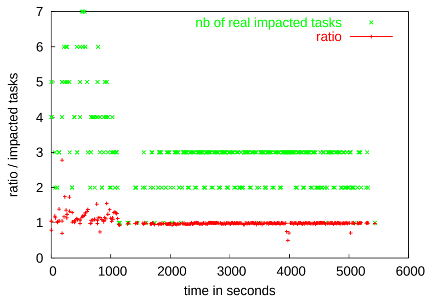

Fig. 2. Scenario (d) Scheduled by MP on Spinnaker

Scheduling decisions rely on the accuracy of the HTM, e.g the ratio of the HTM estimated duration by the real post-mortem duration. We show in this section that it is high but may degrade with the server load and for some types of submission. Two main information appear on Figures 1(1) in dark dots: the value for each task (represented by its submission date on the abscissa) of the HTM estimated duration divided by the real post-mortem one. Hence, the closest to 1 is the ratio, the most accurate the prediction is ; (2) in light dots: the number of tasks that have interfered with the considered task during its execution.

We can observe in Figure 1 the accuracy of the HTM on the 222 tasks which have been scheduled by MSF on artimon. Estimations are degrading with the load of the server. Indeed, until 2800 seconds, estimations are more than 96% accurate on the average and still 93% accurate until 3156 seconds. Then, more than 6 tasks are simultaneously executed on artimon, thus increasing prediction errors. The HTM regains a high accuracy when the load decreases. We can note that the HTM estimations accuracy is not decreasing with time.

We can observe on Figure 2 the information collected on the 301 tasks scheduled by MP on spinnaker, the fastest server, during an experiment of the scenario (d). We can observe two regions: before and after time equals 1000 seconds. Until time reaches 1000 seconds, accuracy is uneven. During that time, the server executes an average of 4 tasks running concurrently, with a peak equal to 7 tasks. After 1000 seconds, a maximum number of 3 tasks are running concurrently. The HTM is over 95% accurate.

As a conclusion, we see that the HTM accuracy is high when the number of simultaneous running jobs is lower than 5. Moreover, for some scenarii ((d) and independent tasks submissions) the HTM is able to regain a high accuracy alone and the accuracy does not degrade with time.

5.3 Comparison Between Heuristics

For each scenario we compare each heuristics on three criteria. MCT is used as the comparison baseline. First, we compare the application completion time, e.g. the *makespan*. This comparison is done for all the scenarii. Second, when independent tasks are involved (scenarii (c), (f), (g) and (h)), we compare the *response time*, e.g. in our context (see section 2), the average duration of a task. The response time is only measured for each independent task. Third, when independent tasks are involved, we give the percentage of independent tasks that finish sooner when the experiment is scheduled with our heuristic than with MCT. If an heuristic is able to give a good response time and a majority of tasks finishing sooner than if scheduled with MCT, it improves the quality of service given to *each client*.

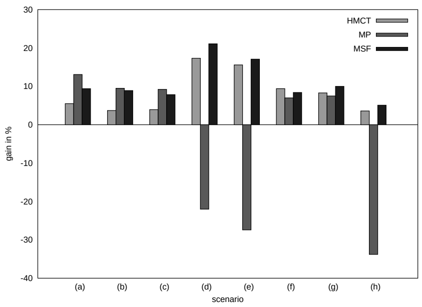

Fig. 3. Gain on the Makespan for each Client of each Scenario

Figure 3 shows the average gain on the makespan performed by each heuristic over MCT *for each client* and for each scenario. We see that MSF and HMCT always outperform MCT up to 22%. MP outperforms HMCT and MSF when applications are 1D-mesh graphs (scenarii (a), (b) and (c)). However, MP shows negative performance for some stencil graphs scenarii. This is explained as follows. Due to its design, MP can unnecessarily map a task on a slow server, for example when it is the only idle one. This delays the completion of the graph because some critical tasks are mapped on slow servers. The client must wait for the completion of these tasks before being able to resume the execution of the application.

Figure 4 shows, for each heuristic, the average gain on the response time that *each independent task* benefits when scheduled with the considered heuristic. Scenarii (a), (b), (d) and (e) are not concerned and only appear to ease the reading. We see that HMCT, MP and MSF outperform MCT for all the scenarii. For scenarii (f) and (g) MSF is the best heuristic with a gain up to 40%. For scenarii (c) and (h) MP is the best heuristic, with more than 67% of gain in scenario (h). For this scenario HMCT shows nearly no gain.

The average percentage of tasks that finish sooner than MCT is given for each heuristic in Figure 5. On the opposite of the two former figures where a gain is observed as soon as the value is positive, the percentage here has to be superior to 50% to express a gain (note that scenarii (a), (b), (d) and (e) are still not concerned here). We see that, as for the response time, HMCT, MP and MSF always outperform MCT. MSF is the best heuristic for scenarii (f) and (g). MP is the best for scenarii (c), (f) and (h), where 90% of the tasks finish sooner than if scheduled with MCT. For this scenario HMCT has the same performance than MCT.

Figure 4 and Figure 5 show together that MP and MSF are able to offer a good quality of service to each independent task. This means that on the average a task scheduled by one of these two heuristics finishes sooner than MCT and the average gain is high.

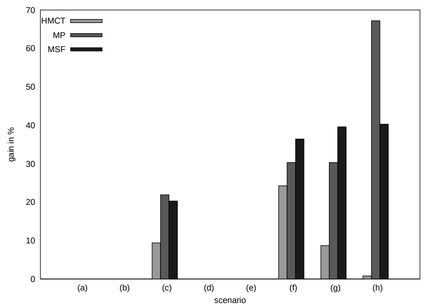

Fig. 4. Gain on the Response Time for each Client of each Scenario

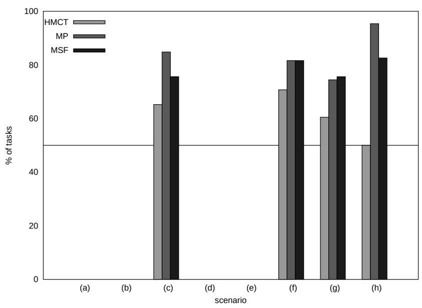

Fig. 5. Percentage of Tasks that Finish Sooner for each Scenario

6 Conclusion

In the paper, we tackle the problem of scheduling tasks in gridRPC systems. MSF, which rely on a non-intrusive predictive module called the Historical Trace Manager. It simulates the execution of tasks on the environment with the time-shared model.

For our tests, we have used NetSolve, a gridRPC environment, in which we have implemented the HTM and all the heuristics. An extensive study has been performed on a real test platform. Numerous scenarii involving different kind of submissions, with different application graphs possibly concurrent to independent tasks, have been submitted to the modified NetSolve agent.

We have shown that the HTM is accurate when the server load is not too high. In some cases, the HTM accuracy degrades during the execution due to error accumulations.

Our heuristics were compared against Minimum Completion Time, the default NetSolve scheduling heuristic.Results show that our heuristics outperform MCT on at least two of the three observed criteria: makespan, response time and percentage of tasks finishing sooner than when scheduled with MCT. MSF appears to be the best overall heuristic as it always outperforms MCT on all the criteria.

In our future work, we will implement a mechanism to synchronize the HTM to the reality. This mechanism will be based on the task completion date. The goal is to limit the effect of error accumulation seen in some experiments. Early developments show an improvement in the HTM predictions accuracy.

References

1. Nakada, H., Matsuoka, S., Seymour, K., Dongarra, J., Lee, C., Casanova, H.: A GridRPC Model and API for End-User Applications (2003) `https://forge.gridforum.org/projects/gridrpc-wg/document/GridRPC_EndUse%r_16dec03/en/1`.
2. Casanova, H., Dongarra, J.: Netsolve : A network server for solving computational science problems. In: Proceedings of Super-Computing -Pittsburg. (1996)
3. Nakada, H., Sato, M., Sekiguchi, S.: Design and implementations of ninf: towards a global computing infrastructure. Future Generation Computing Systems, Metacomputing Issue (1999) 649–658
4. Caron, E., Desprez, F., Fleury, E., Lombard, D., Nicod, J., Quinson, M., Suter, F.: Une approche hirarchique des serveurs de calcul. to appear in Calculateurs Parallles, numro spcial metacomputing (2001) http://www.ens-lyon.fr/ desprez/DIET/index.htm.
5. Maheswaran, M., Ali, S., Siegel, H., Hengsen, D., Freund, R.: Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing system. In: Proceedings of the 8th Heterogeneous Computing Workshop (HCW '99). (1999)
6. Wolski, R., Spring, N., Hayes, J.: The network service : A distributed resource performance forecasting service for metacomputing. Journal of Future Generation Computing Systems (1999) 757–768
7. Baker, K.: Introduction to Sequencing and Scheduling. Wiley (1974)
8. Beaumont, O., Boudet, V., Robert, Y.: The iso-level scheduling heuristic for heterogeneous processors. Technical Report RR-2001-22, LIP, ENS Lyon (2001)
9. Galassi, M., Theiler, J.: The gnu standard library (1996) http://www.gnu.org/software/gsl/gsl.html.