ICPADS'04

July 7-9, 2004, Newport Beach, California

Efficient Scheduling Heuristics for GridRPC Systems

Yves Caniou* Emmanuel Jeannot

LORIA, Université Henri Poincaré, campus scientifique, BP 239, 54506 Vandœuvre lès Nancy, FRANCE

{caniou,ejeannot}@loria.fr

Abstract

In this paper we examine several scheduling heuristics for GridRPC middleware relying on the time-shared model (a server can execute more than one task at a time). Our work is based on a forecast module called the 'Historical Trace Manager' (HTM), which is able to predict durations of tasks in the system. We show that the predictions performed by the HTM are very accurate. The five proposed scheduling heuristics use these predictions to map submitted tasks to servers. Experimental simulation results show that they are able to outperform the well-known MCT heuristic for several metrics (makespan but also sumflow, max-stretch, etc.).

Keywords: time-shared resources, dynamic scheduling heuristics, historical trace manager, GridRPC

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

1. Introduction

Recently a standardization of Remote Procedure Call for GRID environments has been proposed [2]. This standardization defines an API and a model. The model is composed of three parts: clients, servers and an agent (also called a registry): a client contact the agent to access the underlying resources connected to it. Several Problem Solving Environments (PSE) such as NetSolve [CD96], Ninf [HN99] and Diet [CDF+01], instantiate this model. They allow distant and transparent access to distributed resources, usually called *metacomputing*. Other systems are *application-centric*, e.g. they are optimized for an application type that can be characterized (AppleS [BW97] and APST [COBW00]). These middlewares allow parallelism and the access to scientific optimized libraries, in areas as various as biochemistry, fluid mechanic, nuclear ([SBSS98],[NHR85])...

For the execution of an application on a distributed environment to be most effective, it is relevant to optimize the choice of the resources where its composing tasks are mapped. In a PSE, the agent is the focal point charged to optimize the schedule according to a certain metric. If the metric is the application completion time (makespan), mapping independent tasks onto an heterogeneous environment is NP-complete [IK77]. Then, scheduling decisions rely on heuristics.

In the literature, it is often assumed that a server can compute only one task at a time [BSB+99]. In this context, MCT was designed in [MAS+99] to map tasks on servers. It is used in many heterogeneous systems and computes estimated finishing dates assuming that each server executes sequentially submitted tasks. Conversely, some PSE servers such as NetSolve's, execute the task as soon as input data is received. NetSolve also uses

[2] `https://forge.gridforum.org/projects/gridrpc-wg/document/GridRPC_EndUser_16dec03/en/1`

MCT to schedule the tasks. However, since MCT was designed for mono-client system with sequential execution of submitted task on each server, it gives suboptimal results for GridRPC systems when multiple clients can submit tasks and servers can execute multiple tasks at a given time.

In this paper we propose and examine five scheduling heuristics for GridRPC systems. We use the historical trace of already mapped tasks to simulate the environment and take scheduling decisions accordingly. Hence, we have developed the HTM (Historical Trace Manager), a distributed computing environment simulator, where resources (network and servers) can be shared.

Our contribution is twofold. First, we show that the predictions performed by the HTM are very accurate when the load of each server is not too high. This induces the validation of simulation results. Second, we have implemented and tested several scheduling heuristics based on the HTM. We have compared them with MCT on different metrics and show that some of them outperform MCT.

This document is organized as follows: section 2 describes the GridRPC model. We introduce the HTM and the heuristics in section 3. The accuracy of the HTM and the simulation results validity are evaluated in section 4 ; in section 5, we present the metrics on which heuristics performances have been compared ; we explain the experimental work in section 6 and give the results in section 7 ; we then conclude in section 8 and present our future objectives.

2. GridRPC Environment

Our work deals with scheduling tasks in GridRPC environments [SNM+02] instantiating the common client-agent-server model, which we describe here.

2.1. Overview

Some middlewares are available for common use and designed to provide network access to remote computational resources for solving computationally intense scientific problems. Some of them, like Netsolve [CD96] and Ninf [HN99], rely on the GridRPC model. Built on top of GridRPC, the system is usually divided in three parts: clients which need some resources to solve numerous problems, servers which run on machines that have resources to share and an agent that contains the scheduler and maps the requested problems of clients to the available servers. Each machine of such a system can be on a local or geographically distributed heterogeneous computing network.

The submission mechanism works as follows: the client requests the agent for a server that can compute its job. The agent sends back the identity of the server that scores the optimum.

In order to score each server, the scheduler needs the most accurate information on both the problem and the servers (static information) as well as on the system state (dynamic information). Static information concern each server (network and CPU peak performances) and problem descriptions (size of input and output data as well as the task cost: number of operations requested to perform the problem). Dynamic information concerns each server (current CPU load, current bandwidth and latency of the network). How these information are computed depends on the implementation of the middleware and is out of the scope of this paper.

2.2. MCT and the Agent

Minimum Completion Time (MCT) [MAS+99] is used in many middlewares (such as NetSolve) to schedule the tasks on the servers. Its is described in Fig 1. It scores each server according to the estimated finishing date of the remote call. This prediction is computed assuming that the last recorded measures are constant during the execution of the job.

```
For all server S that can resolve the new submitted problem
    D1(S) = estimated amount of time to transfer input and output data.
    D2(S) = estimated amount of time to solve the problem.
Choose the server that scores the optimum, e.g. that minimizes D1(S) + D2(S).
```

For all server S that can resolve the new submitted problem
- $D_1(S)$ = estimated amount of time to transfer input and output data.
- $D_2(S)$ = estimated amount of time to solve the problem.

Choose the server that scores the optimum, e.g. that minimizes $D_1(S) + D_2(S)$.

Figure 1: *MCT algorithm*

MCT was originally designed to minimize the makespan. It is a robust, fast, efficient algorithm when tasks are executed sequentially on servers. In our context, a task starts as soon as input data is completely received. Thus, two or more tasks can compete for the processor(s) of the same server at the same time. Therefore it is needed to redesign an algorithm for this context.

3. Historical Trace Manager and Heuristics

Let us assume that a server can run more than one task at a time (as is implemented in a NetSolve environment). This is the case when the submission rate is high or when the environment is heterogeneous. We have then developed the Historical Trace Manager (HTM), which is called by the agent when a new task arrives. It gives additional relevant information to compute mappings. We also present five heuristics that rely on HTM information.

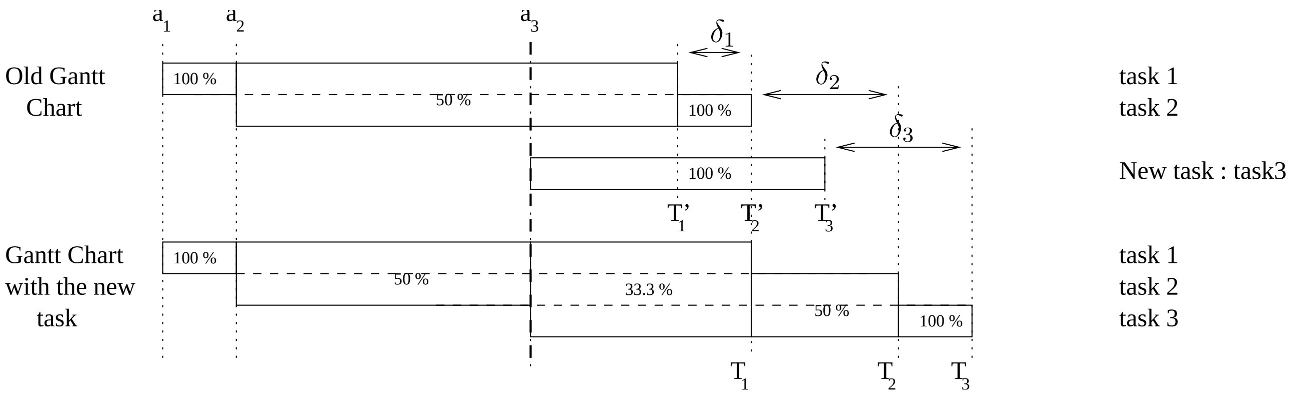

Figure 2: Notations for the Historical Trace Manager

3.1. Notations

We use the following notation: a_i is the arrival date of task i. T'_i is the simulated finishing date in the current system state and C_i is the real one (*post-mortem*). The HTM can simulate the execution of a new task n and give the new simulated completion dates T_i of all tasks $i, i \leq n$. We define for all $k \leq n, \delta_k = T_k - T'_k$, the *perturbation* the task n produces on each running ones (Fig 2). We also define for all $k \leq n, D_k = T_k - a_n$, the *remaining* duration of the task k before completion. $p(i)$ is the server where the task i is mapped and d_i its duration on the unloaded server.

3.2. Overview

The HTM has two goals. First, it records all available information for each task and second, it computes the Gantt chart for each server. The scheduler can therefore compute the impact the new task would have on all the tasks previously mapped on this server and predict all completion dates.

In order to build or update the Gantt chart, the HTM uses the time-shared model: if a server runs n tasks at a given time, then each task is given $1/n$ of the server power. One can see on the top of figure 2 the Gantt chart before the new task submission and the bottom shows the new Gantt chart after the simulation of the new task. Task 1 starts at time a_1 and runs using 100% of the CPU until task 2 starts at time a_2 where the two tasks receive 50% of the CPU. In this system state, task 1 and task 2 are scheduled to finish respectively at time T'_1 and T'_2. Task 3 arrives on the server at time a_3. Without any task running on the server, it should finished at time T'_3. The HTM simulates the execution of this task on the server and computes a new Gantt chart (bottom of the figure). One can see that from a_3 to the first finishing date of a task, here T_1, the three tasks share the CPU and receive 33.3% of its power. The HTM predicts task 3 to complete at T_3.

The simulation is done for every three steps of the execution of a task: input transfer, computation, output transfer. The accuracy of the HTM on a real platform is shown in section 4.

3.3. Utility of the HTM

Let us suppose that the environment contains only 2 *identical* servers (same network capabilities, same CPU speed peak, same set of problems etc.). At time 0, the client sends to the servers two tasks T_1 and T_2, whose durations are 100 and 1000 seconds respectively, with no input data. Suppose that T_1 is scheduled on the server 1 and T_2 on the server 2. At time 80, let a client submit to the agent a task T_3 whose duration is 100 seconds. Without the historical trace manager, the agent only knows that server 1 and server 2 have the same load and therefore is not able to decide which is the best server to schedule T_3. However, the historical trace manager simulates the tasks on each server and the heuristic knows that the remaining duration of task T_1 is 20 seconds while the remaining duration of task T_2 is 920 seconds. Therefore it knows that scheduling T_3 on server 1 will lead to a shorter completion time than scheduling T_3 on server 2.

3.4. Heuristics

The HTM is able to predict the duration a task would take on a server, but can also be used to design heuristics capable to handle other metrics in addition to the makespan. Indeed, we present in this section some heuristics that consider the perturbation which a task induces on others (more are presented in [CJ02]).

- **Historical MCT: HMCT** is MCT using HTM information: it simulates the new task on each server. Then, it selects the server that gives the best finishing date for this task, e.g. the soonest.

- **Advanced HMCT:** The HTM simulates the incoming task. For each server s, there is a task t_s that finishes the latest. **AHMCT** chooses the server s_0 where the last finishing date is the soonest, e.g where $T_{t_{s_0}} = \min_s T_{t_s}$. We expect that minimizing the makespan at each step will help to minimize the makespan at the end.

- **Min Perturbation:** From information given by the HTM after the simulation of the new task n, **MP** chooses the server that minimizes the perturbation $\sum_{i=1}^{n-1} \delta_i$. In case of equality, for instance when the system starts, the server where the new task finishes the soonest is chosen.

- **Min Length:** The HTM simulates the new task n on each server. Then, **ML** chooses the server that minimizes the quantity $\sum_{i=1}^{n} D_i$, e.g. the sum of the remaining quantity of each task on the server at the new task arrival date, including the new one.

- **Min SumFlow:** The HTM simulates the new task and then computes for each server s, $P_s = \sum_i (T_i - a_i)$. **MSF** choses the server s_0 that minimizes P_s. But as the difference between two values is only due to perturbations and to the new simulated task duration, the heuristic only needs to compute $\sum_{i=1}^{n-1} \delta_i + T_n - a_n$ for each server s, that is to say the perturbation of the last task on the server plus the estimated length of the new task. Finally, this heuristic is the same as *MTI* (minimize total interference) proposed by Weissman in [Wei96].

4. Validation of Simulation Results

In this section, we compare real tasks durations against their estimations made by simulation in the HTM. There are two objectives: firstly, to determine the accuracy of the HTM and secondly, to evaluate the limits of simulation results.

In order to perform the following tests, we have integrated our HTM into the code of the agent in NetSolve. We have performed 100 experiments where 500 non identical and independent jobs have been requested. Indeed, a job can require 20, 35 or 50 seconds on the unloaded server. The real and the HTM estimated duration of each task during each experiment have been recorded. Figure 3 show representative results, highlighting two main areas of information:

- In dark dots: the ratio for each task of the HTM estimated completion date divided by the real post-mortem one, indexed by the submission date on the abscissa. Hence, the closest to 1 is the ratio, the most accurate the prediction is ;
- In light dots: the number of tasks that have interfered during the task execution.

Results show that the HTM predicts accurate completion dates of previously assigned and still running tasks, taking into account interferences tasks have on each other. Then HTM information are relevant to give more accuracy to load reports.

However the accuracy is degrading when too many tasks (5 or more) are executed on a server. This occurs when the rate is much too fast for the environment or when the heuristic tends to overload some servers. We should also note that the HTM always predicts tasks flow greater than in reality.

Because the HTM is an environment simulator, results induce that independent tasks simulations are relevant to foresee what can be expected in reality. Indeed, the HTM and the Simgrid tool [Cas01] for example use similar simulation mechanisms. Nonetheless, there are some limitations due to the arrival rate, to the heterogeneity (tasks and servers) and to the heuristic: accuracy is obtained if less than 5 tasks are executed simultaneously on a server.

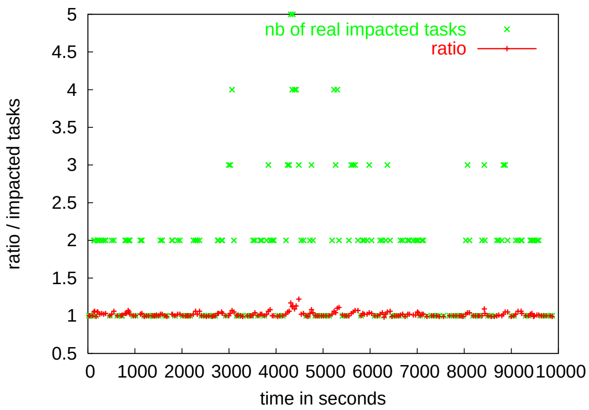

Figure 3: Ratio of the HTM Estimated Duration by the Real Post-Mortem Duration of Tasks of a Set of 500 Tasks Submission

5. Performance Metrics

In this section, we present metrics that have been observed when comparing our heuristics against a modelization of MCT. In gridRPC middleware it is important to improve the speed of the application but also the ressources utilization and the quality of service for each requests. This is why we do not only observe the makespan metric.

- **Makespan**: It is the completion time of the last finished task, $\max_i C_i$. Minimizing the makespan is usually the goal of the schedule of an application: the sooner the application is finished, the greater it is. Nonetheless, the makespan of a set of tasks is mainly due to the last completed task starting time [MAS+99, CJ02] and because independent tasks can be submitted by several clients, this may not be the main metric to optimize here.

- **Sum-Flow** [Bak74]: This is the amount of time that the completion of all tasks has taken on all resources, $\sum_i (C_i - a_i)$. Executing tasks on servers can have a cost proportional to their duration, the cost possibly being a function of servers computing power.

- **Max-Flow** [lABCM98]: It is the maximum amount of time a task has spent in the system, $\max_i (C_i - a_i)$.

- **Max-Stretch** [lABCM98]: The stretch of a task i is defined by $s_i = (C_i - a_i)/d_i$. It is the factor by which a task has been slowed down relative to the time it takes on the same but unloaded server. The *max-stretch*, defined as $\max_i s_i$ is by what maximum factor a task has been slowed down relative to the time it takes on the unloaded server. This can represent a *Quality Of Service* of the scheduler.

- **Number of tasks that have finished sooner**: Whereas this is not a metric, this value gives in correlation with the previous metrics a relevant idea of a *quality of service* given to each independent task *when comparing two heuristics*. For instance, comparing the heuristics H_1 with MCT (on the *same* set of tasks $\{t_1 \dots t_n\}$ and *same* environment), it is $\mid \{i | C_{iH_1} < C_{iMCT}\} \mid$.

The user point of view is not that the last submitted task finishes the soonest (makespan policy) or benefits the most of the system, but that his own tasks (a subset of all client requests) finish as quickly as possible. Therefore, if we can provide a heuristic where most of the tasks finish sooner than with MCT without delaying other tasks completion date (that can be verified with the sumflow for example), we can then claim that this heuristic, for the user, outperforms MCT.

6. Experimental Work

A Problem Solver Environment is simulated with the Simgrid tool [Cas01]. The dynamic mapping heuristics are evaluated using some parameters that characterize heterogeneous servers and each task of the metatask. We explain in this section the different models we used for client-agent-server mechanisms, heterogeneous entities and independent tasks. We used the Gnu Standard Library [GSL] for all the probabilistic distributions used thereafter.

We assume the agent has perfect knowledge of the following information:

- current server load ;
- current network load ;
- peak CPU and network performances ;

- number of operations of any tasks ;

- size of the input and output data of any tasks.

MCT needs all these information while the historical trace manager (and in consequence, our heuristics) needs only the static ones (last three items). Static information are easier to compute accurately as compared to dynamic (for example, tasks durations can be obtained from benchmarks or from means of executions performed on each server [Qui02]). Therefore, in our simulation, the performance MCT will obtain using these information will be better than in any real GridRPC middleware.

6.1. Platform Model Characterization

We assume the client and the agent are able to reach each server. The experiments are conducted on the basis that servers are dedicated to the environment (for instance a set of clusters where reservation is possible). Moreover, we suppose that all problems can be solved on any server. We do not considered any arrival/withdrawal of any server in the system.

6.2. Application Model Characterization

We consider independent task submissions (tasks that have no precedence relation), requested by one or more users. A task is not preemptible: it can not be stopped and continue later nor be removed to be scheduled on another server.

A task begins to be executed on a server as soon as all of the input data has been sent from the client. We consider that a task is finished when the data output is completely received by the client.

6.3. Instantiation

We want to test how the heuristics react under different environment conditions and we want to achieve the best possible overview when comparing each of them against MCT. Therefore, same environments and same task sets are generated for each heuristic: comparisons are conducted on the same sets of tuples (servers, tasks, arrival dates, etc.).

Experiments are conducted with a number of servers held to 25. They have one processor and their computing capacity is drawn from a uniform distribution in the range $[150000, 500000]$. Their network cards have the same performance (100Mbits/sec).

We use a uniform distribution to draw the input and output sizes of data to be transferred between the client and the server from the range $[1Ko, 300Mo]$. The computation cost is generated from a uniform distribution, with the following rules:

- the computation phase costs more than 10 times than the transfer phase;
- the computation phase must not be greater than 600 seconds on the fastest server of the 25 available.

To draw task arrival dates, we use a Poisson distribution whose parameter μ varies from 0 to 70. Simulation results give an asymptotic point of view for $\mu < 10$: for these rates, MCT schedules more than five tasks on a server, and results are consequently distorted. On the other hand, when $\mu = 70$, each server is executing at most one task at a given moment for half of the heuristics tested. For each value of μ, we have varied the number of tasks from 10 to 250. Each plot in a graph is the mean of results of 1000 different tasks sets: each heuristic required 200000 simulations.

7. Results

Results are given in figures 4,5,6,7 and 8. Each is composed of five 3D graphics, one for each observed metric. They present the gain in percentage of the heuristic over MCT. For example, the first graph shows the percentage of 'tasks that finish sooner'. It shows a gain if the value is greater than 50. In the others, a gain is performed as soon as the percentage is positive.

Considering that the makespan of an application composed of independent tasks is mainly due to the last completed task starting time [MAS+99, CJ02], we cannot expect *a priori* a great gain on the makespan.

To compute servers scores in our simulations, MCT uses information which are far more precise than in a real environment (NetSolve). In consequence, our modelization of MCT behaves better than the algorithm does in reality. Therefore, if we build a heuristic that outperforms our simulation of MCT, this heuristic will certainly outperform MCT in reality.

7.1. HMCT

Figure 4 shows that there is a gain greater than 8% on the makespan for $\mu \leq 20$. For $\mu \geq 30$ the gain is still positive even if almost null. The Sum-Flow is greater for HMCT than MCT for $\mu > 10$ (leading to negatives performances). Because HMCT tends to optimize the use of fast servers for new tasks, it delays the running ones (for $\mu \leq 10$, the rate is so high that the flow is increased for both MCT and HMCT). It also results in a percentage of 'tasks that finish sooner' lower than 50% (under 20% for $\mu = 30$). For $\mu \geq 40$, the Max-stretch shows that a task is 30% longer than if scheduled with MCT: there are interferences on fast servers, even at a low rate.

HMCT has a drawback: it tends to overload fast servers but real-world servers cannot handle too many jobs. Consequently, when using HMCT in a high heterogeneous network, there is a risk for faster servers to collapse.

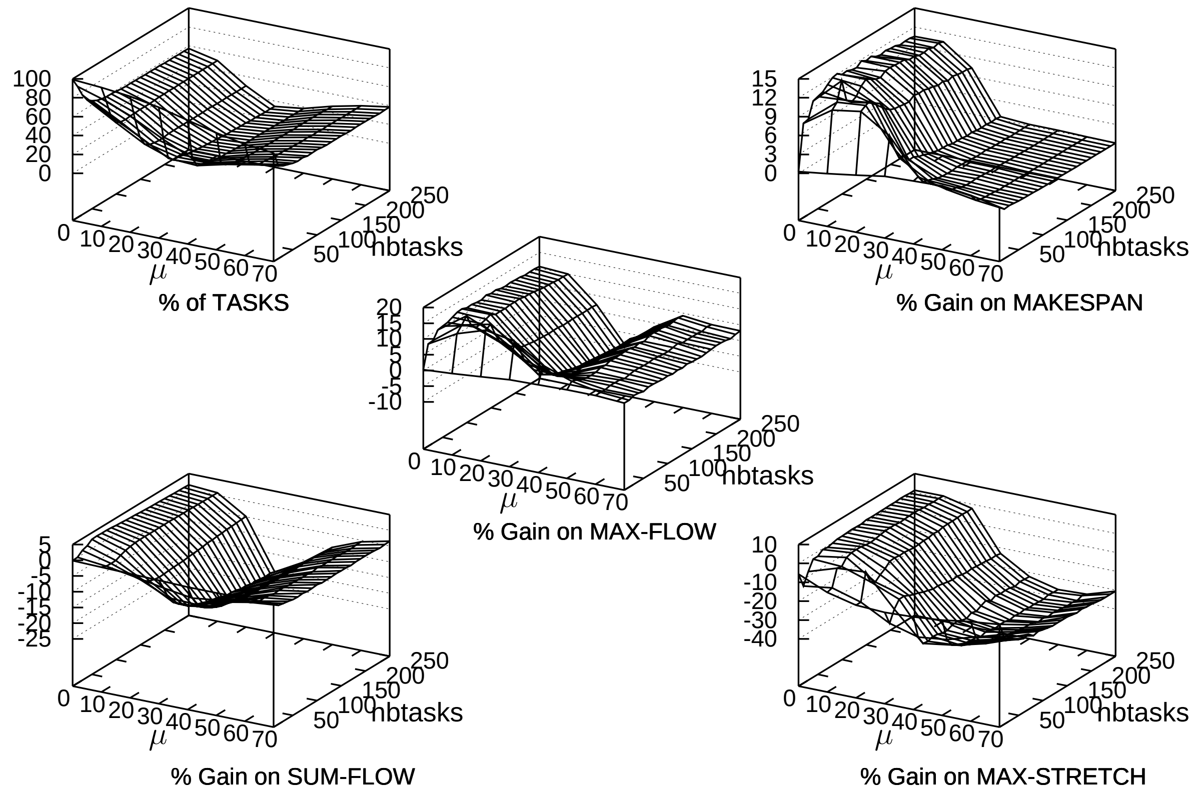

Figure 4: Results for HMCT vs. MCT on 25 servers

7.2. AHMCT

One can see that AHMCT behaves mostly like HMCT (Fig 5). The main differences concern the max-stretch, where it gives better results against MCT than HMCT for $\mu \leq 30$, and the percentage of tasks that finish sooner, where it is always outperformed by HMCT.

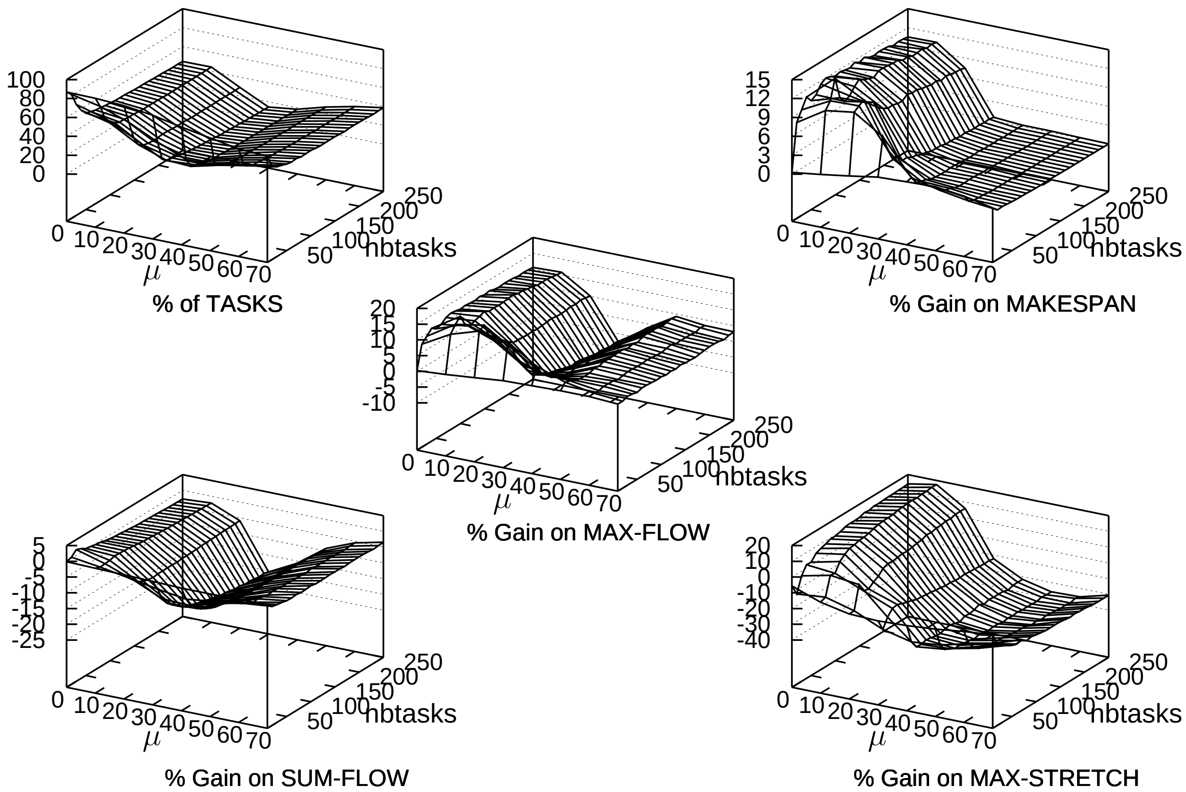

Figure 5: Results for AHMCT vs. MCT on 25 servers

7.3. Min Perturbation: MP

The figure 6 shows that when $nbtask > 80$, the gain on the makespan is positive (greater than 5% for $nbtask = 250$ and $\mu < 30$). MP allows gain on the Sum-Flow, with a peak at 15% for $\mu = 30$. The percentage of 'tasks that finish sooner' is always greater than 60%, with a peak at 70% for $\mu = 30$, (when in most cases, no more than one task is running on a given server). There is always a gain on the Max-Stretch: in some cases, the gain reaches 90%. The Max-Flow is always better except for $\mu = 30$.

Despite these good results, MP presents a major drawback: when only one server is idle, it is chosen regardless its speed, possibly jeopardizing its performance. This happens when dealing with highly heterogeneous resources and/or a high rate of costly requests: for

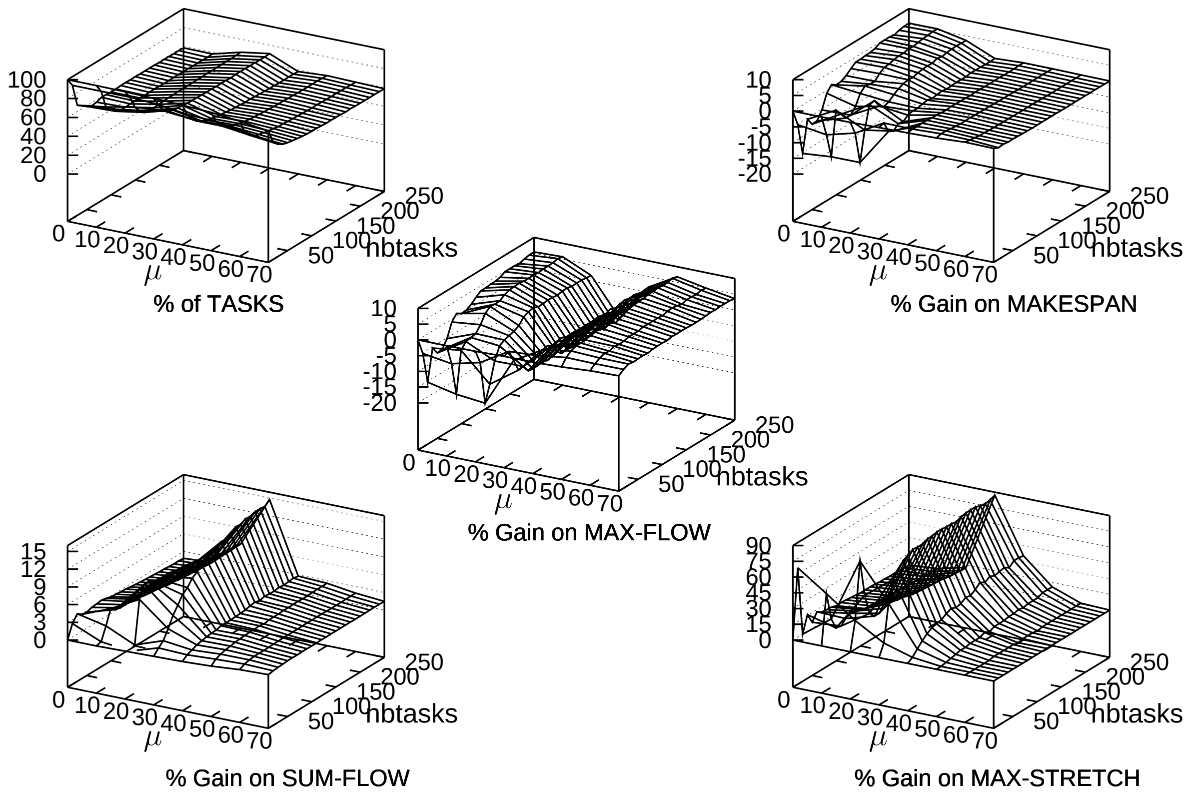

Figure 6: Results for Min Perturbation vs. MCT on 25 servers

example, when $\mu < 30$ and $nbtasks < 50$, MP is outperformed by MCT on the makespan.

7.4. Min Length: ML

Except for $\mu = 0$, the gain on the makespan is always positive (Fig 7). ML has also a positive gain on the Sum-Flow, with a peak at 15% when $\mu = 10$. On the Max-Stretch, MCT performs slightly better for $20 \leq \mu \leq 40$ and gives around 60% on the percentage of 'tasks that finish sooner', with a peak at 80% for $\mu = 50$.

ML achieves a makespan at least as good as MCT for $\mu \neq 0$, an adequate performance on the Max-Stretch and gains on the Sum-Flow, the Max-Flow and on the percentage of 'tasks that finish sooner'. Since the mapping decision is partly taken using the cost of the new task, ML does not present MP's drawback. Therefore, this heuristic tends to overcome

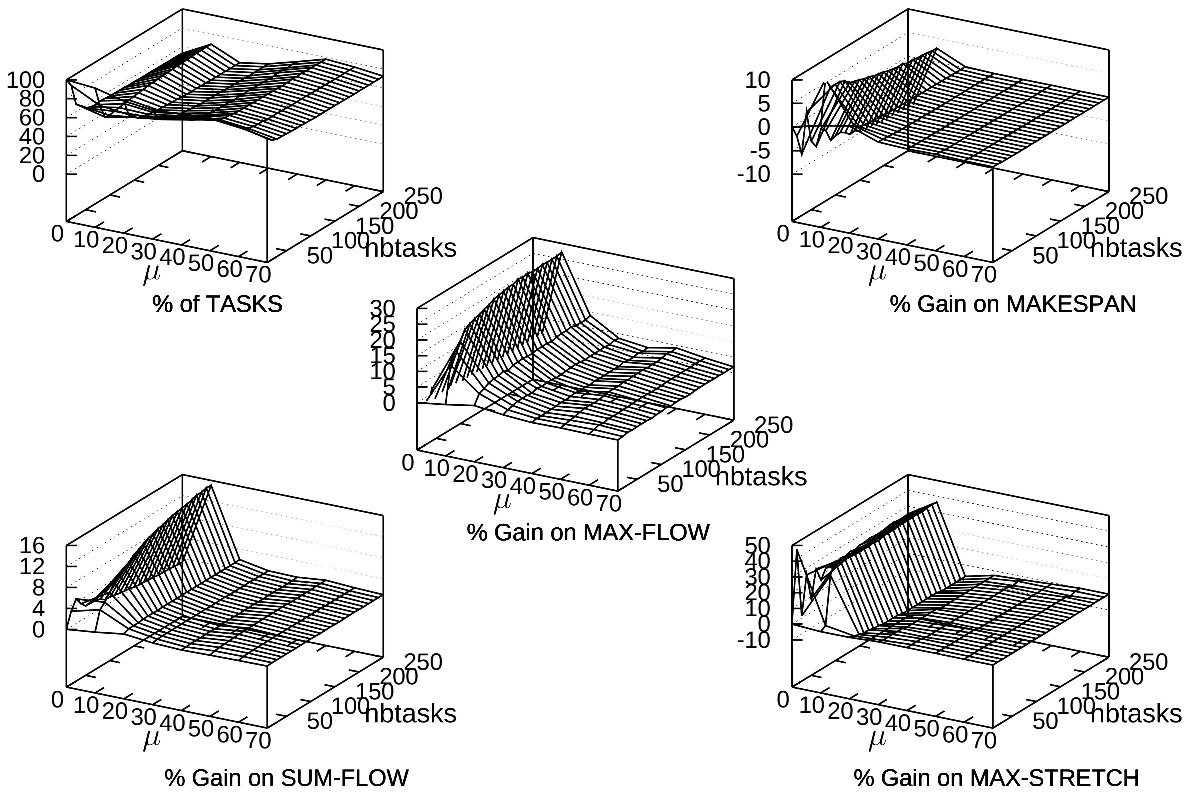

Figure 7: Results for Min Length vs. MCT on 25 servers

the drawbacks of HMCT and MP, while keeping the advantages of both.

7.5. Minimum Sum Flow

MSF presents great performances on the makespan, around 9% when $\mu \leq 20$, against MCT (Fig 8). The percentage of 'tasks than finish sooner' is slightly worse than MCT's for $25 \leq \mu \leq 50$, but is always above 40%. For $\mu \geq 35$, tasks are slowed down by a maximum factor of 20% (occuring when $50 \leq \mu \leq 60$). Sum-flow performances depend on the parameter μ: MSF maximizes the use of fast servers when $30 \leq \mu \leq 40$ and so delays tasks. It induces a greater flow cost. But when $\mu \leq 30$, MCT overloads faster servers and MSF, taking into account interferences, performs slightly better than MCT.

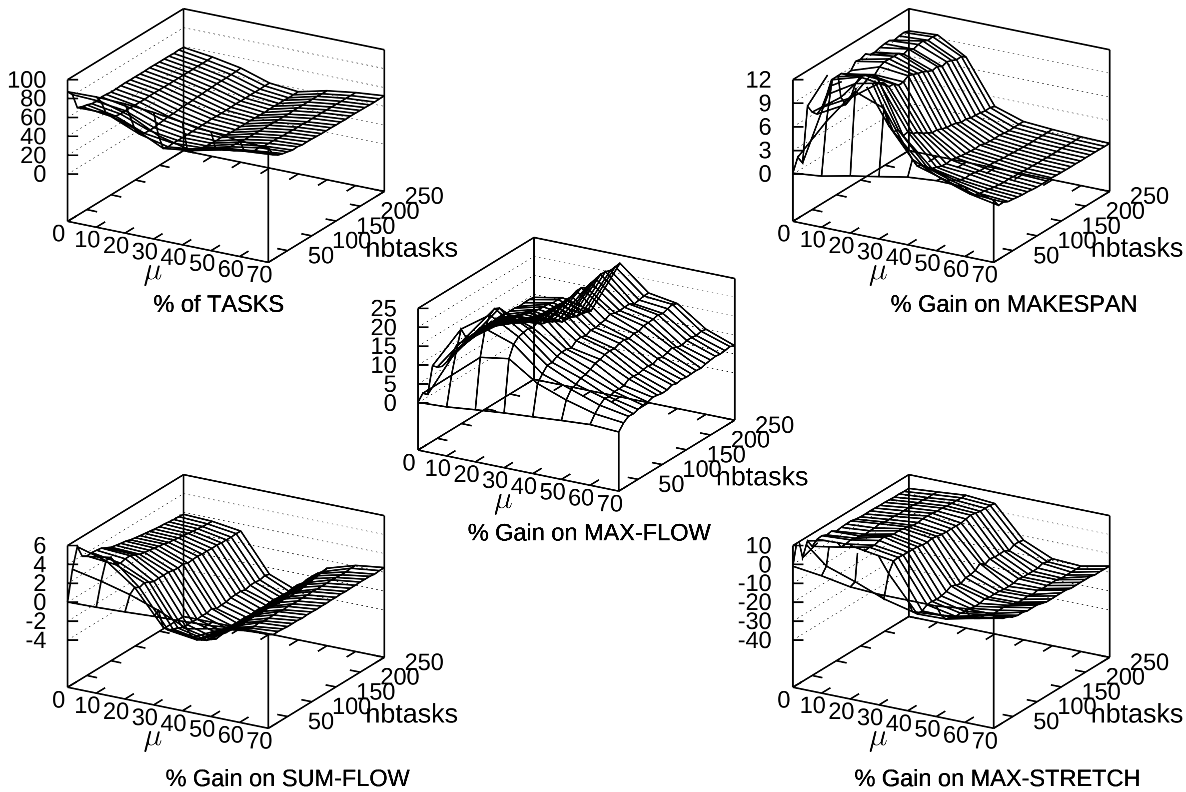

Figure 8: Results for MSF vs. MCT on 25 servers

7.6. Discussion

To summarize our heuristics performances, we can conclude that our heuristics outperform MCT on the makespan. However, the percentage of 'tasks that finish sooner' of HMCT is poor and the Max-Stretch is always negative. In a client-agent-server context, this needs to be improved. To what we expected, AHMCT gives almost identical results as HMCT, with less tasks 'finishing sooner' but a better performance on the max-stretch. MP shows great performances on every metrics but can present the drawback to unnecessarily utilize slow servers. ML tends to overcome HMCT and MP drawbacks in addition to good performances on every metrics, except when all tasks are submitted at the same time (e.g. $\mu = 0$, an almost impossible situation). MSF behaves better than MCT. Indeed, it combines HMCT and MP performances. It outperforms ML on the makespan but looses on the other metrics.

Since ML has positive results on all observed metrics and outperform all the other heuristics on the percentage of 'tasks that finish sooner than MCT', we consider that overall it is the best one among all the tested heuristics in this paper.

8. Conclusion and Future Work

GridRPC is an emerging standard for metacomputing middleware. Therefore, it is important to design efficient scheduling heuristics for this model.

Consequently, we have presented the Historical Trace Manager, a predictive and recording module that can be used in Problem Solver Environment relying on the GridRPC model. We have shown that it is able to give relevant and accurate information on the estimated duration of any task in the environment if servers are not overloaded. This leads *to validate the use of time-shared simulations of independent tasks to study new heuristics*.

Five scheduling heuristics that rely on HTM information have also been presented: Historical MCT, Advanced HMCT, Min Perturbation, Min Length and Min Sum Flow. With the availability of HTM information, they are able to compute the perturbation which tasks induce on each other. We have performed an extensive simulation study on five metrics and various parameters. They show greater performances against the modelization of MCT, which has a far better understanding of the environment than in reality. Thus, they are all expected to outperform MCT in real solving environment. Nonetheless, HMCT, AHMCT and MP may not be the best heuristics in a production environment where no information is available on the rate of submission for example. Results show that MSF and ML outperform MCT, but ML gives a better quality of service, ML is then the best heuristic.

We will now aim to see how heuristics behaves in a real environment, improve the reliability of the HTM predictions with a message sent to the agent when a task finishes and consider memory requirements.

Even if the agent decision is short (under 10 msec), the scalability may be a problem. We plan to build a scalable version of the HTM in order to distribute the scheduler in the DIET environment, which has a hierarchy of agents [CDF+01].

References

[Bak74] K.R. Baker, *Introduction to sequencing and scheduling*, 1974.

[BSB+99] Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau Bölöni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, Bin Yao, Debra Hengsen et Richard F. Freund, *A comparison study of static mapping heuritsics for a class of meta-tasks on heterogeneous computing systems*, Proceedings of the 8th Heterogeneous Computing Workshop (HCW '99), april 1999.

[BW97] F. Berman et R. Wolski, *The apples project : A status report*, Proceedings of the 8th NEC Research Symposium, may 1997, http://apples.uscd.edu.

[Cas01] H. Casanova, *Simgrid: A toolkit for the simulation of application scheduling*, Proceedings of the IEEE Symposium on Cluster Computing and the Grid (CCGrid'01), IEEE Computer Society, may 2001, available on http://www-cse.ucsd.edu/ casanova/.

[CD96] Henry Casanova et Jack Dongarra, *Netsolve : A network server for solving computational science problems*, Proceedings of Super-Computing -Pittsburg, 1996.

[CDF+01] E. Caron, F. Desprez, E. Fleury, D. Lombard, J.M. Nicod, M. Quinson et F. Suter, *Une approche hiérarchique des serveurs de calcul*, to appear in Calculateurs Parallèles, numéro spécial metacomputing (2001), http://www.ens-lyon.fr/ desprez/DIET/index.htm.

[CJ02] Y. Caniou et E. Jeannot, *Dynamic mapping of a metatask on the grid: Historical trace, minimum perturbation and minimum length heuristics*, Tech. Report 4620, LORIA, nancy, oct 2002.

[COBW00] Henri Casanova, Graziano Obertelli, F. Berman et R. Wolski, *The apples parameter sweep template : User-level middleware for the grid*, Proceedings of the Super Computing Conference (SC'2000), 2000.

[GSL] *The gnu standard library*, http://www.gnu.org/software/gsl/gsl.html.

[HN99] Satoshi Sekiguchi Hidemoto Nakada, Mitsuhisa Sato, *Design and implementations of ninf: towards a global computing infrastructure*, Future Generation Computing Systems, Metacomputing Issue **15** (1999), 649–658.

[IK77] O.H. Ibarra et C.E. Kim, *Heuristic algorithms for scheduling independent tasks on nonidentical processors*, Journal of the ACM **24** (1977), no. 2, 280–289.

[lABCM98] Michæl A. Bender, Soumen Chakrabarti et S. Muthukrishnan, *Flow and stretch metrics for scheduling continuous job streams*, SODA: ACM-SIAM Symposium on Discrete Algorithms, 1998.

[MAS+99] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hengsen et Richard F. Freund, *Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing system*, Proceedings of the 8th Heterogeneous Computing Workshop (HCW '99), april 1999.

[NHR85] W.R. Nelson, H. Hirayama et D.W.O Rogers, *The egs4 code system*, Tech. report, Stanford Linear Accelerator Center Report SLAC-265, 1985.

[Qui02] Martin Quinson, *Dynamic performance forecasting for network-enabled servers in a metacomputing environment*, International Workshop on Performance Modeling, Evaluation and Optimization of Parallel and Distributed Systems (PMEO-PDS'O2), april 15-19 2002.

[SBSS98] J. Stiles, T. Bartol, E. Salpeter et M. Salpeter, *Monte carlo simulation of neuromuscular transmitter release using mcell, a general simulator of cellular physiological processes*, Computational Neuroscience (1998), 279–284.

[SNM+02] Keith Seymour, Hidemoto Nakada, Satoshi Matsuoka, Jack Dongarra, Craig Lee et Henri Casanova, *Overview of gridrpc: A remote procedure call api for grid computing*, Grid Computing - Grid 2002, LNCS 2536, november 2002, pp. 274–278.

[Wei96] J.B. Weissman, *The interference paradigm for network job scheduling*, Proceedings of the 10th International Parallel Processing Symposium, HCW, 1996.