Scheduling on the Grid : Historical Trace and Dynamic Heuristics

Yves Caniou, Emmanuel Jeannot

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Scheduling on the Grid : Historical Trace and Dynamic Heuristics

Yves Caniou — Emmanuel Jeannot

N° 4620

Novembre 2002

THÈME 1

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--4620--FR+ENG

Scheduling on the Grid : Historical Trace and Dynamic Heuristics

Yves Caniou* , Emmanuel Jeannot

Thème 1 — Réseaux et systèmes
Projets Resedas

Rapport de recherche n° 4620 — Novembre 2002 — 34 pages

Abstract: We present a historical trace manager and new dynamic scheduling heuristics that can be used, and are studied, in the client-agent-server model on the '*grid*'. These heuristics rely on the common acknowledgment of the characteristics of the tasks submitted to the agent, but also on the construction of the underlying historical trace of the different tasks submitted to each server. We study each heuristic and compare them on several metrics to an instantiation of MCT (*Minimum Completion time*), chosen as reference heuristic. The simulation experiments we have conducted show that they are likely to give good results when tested in a real environment.

Key-words: time-shared resources, dynamic scheduling heuristics, historical trace manager, MCT, NetSolve

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

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Ordonnancement sur la grille : historique des tâches et heuristiques dynamiques

Résumé : Nous présentons un gestionnaire d’historique des tâches, ainsi que de nouvelles heuristiques qui sont utilisées et présentées dans le modèle ’client-agent-serveur’ sur la grille. Ces heuristiques reposent sur les caractéristiques connues des tâches soumises à l’agent, ainsi que sur la construction en parallèle d’un historique des tâches soumises à chaque serveur. Nous les étudions en les comparant sur plusieurs métriques à une instanciation de MCT. Les résultats d’expérimentation obtenus montrent qu’elles sont à même de donner de bonnes performances dans un environnement réel.

Mots-clés : ressources temps partagées, heuristiques dynamiques d’ordonnancement, gestionnaire d’historique des tâches, MCT, NetSolve

1 Introduction

The client-agent-server model is often used in grid applications -*grid middleware* -. Anyone who can contact an agent can have access to the underlying resources connected to this agent. It is transparent for the user, whereas these resources are distant and distributed on the grid, that we usually call metacomputing. Some environments, called **NES** for Network Enable Server, exist, rely on and popularize this model (NetSolve [CD96], Ninf [HN99], Diet [CDF+01]). Some of them are *application-centric*, e.g. they are optimized for an application type that can be characterized (AppleS [BW97] and APST [COBW00] for an application composed of independent tasks). Scientists use them to access scientific optimized functions libraries, in area as various as biochemistry, fluid mechanic, nuclear ([SBSS98],[NHR85])...

For the execution of an application on a distributed environment to be the most effective, it is relevant to optimize the choice of the resources where its composing tasks are mapped. The agent is the focal point that is in charge to optimize the schedule according to a certain metric. In order to determine the network state and the load of each server needed to allocate the task, the agent disposes of its own monitors or eventually uses supervisors beforehand installed such as NWS [WSH99].

In the literature, it is often assumed that a server can compute only one task at a time ([BSB+99],[MAS+99]). We propose here to consider a scheduling in a shared-time but dedicated resources context, e.g. a server loaded with already mapped and running tasks can be chosen to receive new jobs.

To achieve our goal, we consider the historical trace of the already mapped tasks to simulate the environment and take scheduling decisions accordingly. Hence, we have developed a distributed computing environment simulator that takes into account that the network and servers can share their resources with several applications. We have tested in simulated experiments several heuristics. The proposed heuristics, that use the historical trace, are compared to an instantiation of MCT, that models the NetSolve scheduling strategy. The goal of our new dynamic heuristics is not only to optimize the makespan, that is much more an application metric, but also to optimize the finishing time date of each task, that is to disturb the less possible the system state but still be the best for the new incoming task. Moreover, several other metrics, like the makespan and the sum-flow, are considered in this paper.

Our contribution is to consider the historical trace and new dynamic heuristics built on its top. Our proposed heuristics are likely to outperform the MCT algorithm implemented in the real NetSolve system, even on the makespan.

The rest of this document is organized as follow : the section 2 describes NetSolve, an NES relying on the client-agent-server model ; models used for applications and for the grid are defined in section 3 and the management of the historical trace is explained in section 4 ; in the section 5, we show the observed metrics on which we compare the heuristics exposed in the section 6 ; finally we conclude in the section 7 and present our future objectives.

2 NetSolve

We present in this section a **NES** (*Network Enable Server*) called NetSolve [CD96] that uses in its agent body MCT (section 2.3) as a scheduling heuristic.

2.1 Overview

NetSolve [CD96] is a client-agent-server based environment developed in the University of Tennessee, designed to provide network access to remote computational resources for solving computationally intense scientific problems. A NetSolve system consists in three parts : clients which need some resources to solve some problems, servers which run on machines that have some resources available and an agent which maps the requested problems of clients to servers.

The system works as follow : the client requests the agent for a server that can compute its job. The agent uses the information provided to compute its *best* choice. Then, the client requests the answered server. Every machine in a NetSolve system runs a NetSolve server to give access to installed and optimized scientific packages. The client can perform its requests in a blocking or non-blocking fashion.

2.2 NetSolve Model

To choose the *best* server capable to compute the new submitted task, the agent uses the following information :

1. the peek performance of the server, given in Kflops by the LINPACK benchmark ;
2. the load of the server, a number given by the Unix command `uptime` sent sporadically ;
3. the network state. It is the last known bandwidth and latency between the agent and the server ;
4. the parameters and the result data size. It is the data quantity of the input and output that is to be transferred between the client and the server ;
5. the number of operations, requested by the task.

```
For all server $S_i$ that can resolve the new submitted problem
    $D_1(S_i)$ = estimated amount of time for the transfer of the data.
    $D_2(S_i)$ = estimated amount of time to solve the problem.
    Affect the server $S_i$ the score $D_1(S_i) + D_2(S_i)$.
Choose the server $i_0|S_{i_0} = \min_{servers} S_i$.
```

Figure 1: *MCT algorithm*

2.3 Mapping Schema

NetSolve algorithm is described fig 1. To score each server, the agent adds the amount of time that the task needs in its transfer phase to the one in its computing phase, if mapped to the considered server. The score the lowest gives the best server to choose. Hence, the heuristic employed is MCT (Minimum Completion Time) : the best server minimizes the task finishing date. MCT is a robust, rapid algorithm, that gives good results in minimizing the makespan, but it needs accurate information. The refreshment of the measures in the MCT algorithm are sporadic:

- The server load is transferred at least every 60 seconds (if it has varied of more than 20 from the last load), and at most every 300 seconds ;
- the latency and the bandwidth values, once communicated, are refreshed every 1800 seconds ;
- the server notifies the agent the finishing of a task when this occurs.

Moreover, the MCT heuristic is completed by two other mechanisms :

- the client will normally contact the answered server, so the agent adds 100 to the server last recorded workload to correct the load this new task will generate ;
- when receiving a task end notification, if the last load was upper than 100, the server decreases the value by 99.

This two mechanisms are interesting to still try to do good quality choices if new tasks arrive immediately after having mapped a task. But still, the quality of MCT decisions is not necessary the best because of the frequency and the quality of the incoming information on the state of the system. Indeed, on Unix systems, the `uptime` command gives only the average load of the last minute.

The execution table 1 shows the behavior of NetSolve. At time 34:25, the agent increases the load value by 100 due to the first mechanism. But its effect is limited : the server who

time	load/(action)	time	load/(action)
34:25	0 $(send_1)$	39:43	196
34:43	28	41:36	$(receive_1)$
35:43	73	42:43	130
36:35	$(send_2)$	44:26	$(receive_2)$
36:43	121	44:43	75
37:43	174	45:43	28

Table 1: Execution of 2 dgemm 1500 with NetSolve on one server

has recorded an increase of its load greater than 20 sends its new load to the agent at time 34:43. The same happens when the second task arrives at 35:35 with the transfer of the load at time 35:43. The second mechanism has worked but not by itself and its effect is limited : the agent last load value is decreased from 196 to 97 but then refreshed to 130. The second mechanism effect is out when the second task finished, with the load of 130 decreased to 31 at time 44:26 then refreshed at time 44:43 to 75. Therefore, in this example, most of the time the load has a value that does not represent the facts.

Thus, even if the server tells the agent its load immediately after starting its new task, the load would not tell anything useful to the agent. Moreover, if the server communicates its load immediately after the first mechanism has taken place, this value is replaced by the new one which does not reflect what has been done, e.g. the assignation of a task to the server and the acknowledgment the server load will increase. The second point is that even if the mechanisms correct the CPU value during some time, this correction is not really precise. The second mechanism can be of no worth if there was only one task on the server (the load can be less than 100). Third and last point, the estimation of the amount of time of a task is done at a given time, using the last system state recorded, maybe modified, to that time. Nevertheless, the CPU load of a server can decrease in a significant way after the agent has mapped the task, because of the ending of some tasks, leading to choices that would have been better if considering that server.

This is why we propose to build an history of the tasks that are given to servers. The agent knows information about each task allocated to each server and thus can predict when a server will deserve any attention even if it is loaded, and eventually give it some new task to solve.

3 Simulation Model

During the experiments, the network and the different entities that compose the NetSolve system were simulated with the Simgrid tool [Cas01]. The dynamic mapping heuristics were evaluated using some parameters that characterize heterogeneous servers and each task of the metatask. We explain in this section the different models used to refer and model the client-agent-server mechanisms, the heterogeneous entities and metatask, according to the Purdue taxonomy [BSB+98]. We used the Gnu Standard Library [GSL] for all the probabilistic distributions used thereafter.

3.1 Platform Model Characterization

We assume the client is able to reach each server as is the agent. The experiments were conducted on the basis that we are on a dedicated area, that is only the agent can make use of the servers. It is common on clusters using reservation mechanisms (`qsub`). We suppose that we know for each server an accurate estimation of the number of processors, their speed and the network bandwidth.

Moreover, we suppose that all the problems can be solved on any server, as this is not really important for the experiment (that is, when a task arrives, all servers are eligible). We have not tested what would arrive if a server was suddenly struck down, for clients who had jobs on it would have to re-submit their jobs. Likewise, we do not have considered any arrival of a new server in the system.

3.2 Application Model Characterization

We are for the moment interested in metatask mapping, e.g the application is composed of independent tasks (there is no communication between the tasks). For example, we can consider a metatask as composed of Monte Carlo simulations requested by one or more users. It is not in our concern here to deal with the deadline time nor do we consider the preemptive aspect of the problem : a task, if assigned, can not be stopped and continue later nor be removed and be scheduled to another server. If a task has to be duplicated, it is up to the client to do so, requesting twice the agent services.

A task can be executed by a server when all of the input data are finished to be sent from the client. A task is finished when the data output is finished to be sent back to the client. We assume we know an accurate estimation of the size of the input and output data, and the computation cost.

3.3 Instantiation

The assumption of these whole parameters is commonly made when studying mapping heuristics in Heterogeneous Computing Network, and are obtained from benchmarks executed on each server ([Qui02]).

We have chosen the servers cards to be exactly the same (100 Mbits/sec) and the experiments were conducted with a number of servers held to 25. We used a Uniform distribution to generate the heterogeneous set of servers, each of them of one processor A processor can produce a number in the range of 150-500 Kflops.

For describing the dates of arrival of the tasks, we used a Poisson distribution whose parameter μ varies from 0 to 70 (section 6). We used a Uniform distribution to express the input and output size of data to be transferred between the client and the server. The boundary limits are 1 Ko and 300 Mo. The computation cost is generated from a Uniform distribution, with these 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

4 Historical Trace Manager

We have developed a historical trace manager which is sought by the agent when a new task arrives. Its information are at the origin of the heuristics that we will describe later.

4.1 Overview

The historical trace manager has two goals. It keeps the information of each task and computes the Gantt Chart for each server considering the new submitted task. Hence the scheduler can see the impact the new task has on each task previously mapped on this server and predict the completion dates.

As clients are able to reach each server, the network is reduced in our model to the bandwidth of each server full duplex network card. The simulation program allows concurrent sends as well as concurrent receives. Like the computation phase, that begins with the end of the input transfer, they are implemented in a time slice model : n tasks in the same phase on the same server are assumed to take $1/n$ of the server power until one of them finishes. The same schema is repeated until all tasks simulated completion date are obtained. We consider in our model that a context swap takes no time. The figure 2 details an example of a four tasks metatask, with no input and ouput datas, mapped on a server.

We can easily predict the finishing time of all the tasks assigned to each server. Hence, the use of the historical trace when a new task arrives can lead us to consider servers that would not have been otherwise : even if a server is loaded when a new task arrives, its load can decrease later because of finishing tasks. We use these information for new heuristics proposed further.

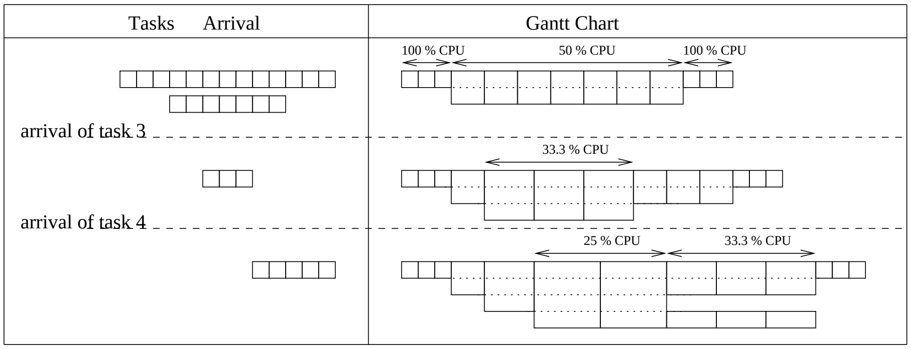


Figure 2: Gantt Charts drawn with the HTM results

size of the square matrix	memory need (Mo)		phase	time needed on server
	input	output		
1200	21.97	10.98	input data cost computing cost output data cost	3 18 1
1500	34.33	17.16	input data cost computing cost output data cost	5 33 1
1800	49.43	24.72	input data cost computing cost output data cost	8 53 2

Table 2: tasks needs

task	arrival date	size of the matrix	real completion date	simulated completion date	error on real duration
1	33.00	1500	80.79	79.99	1.67
2	59.92	1200	92.08	93.19	3.45
3	73.92	1800	142.79	142.50	0.42
1	29.41	1500	76.69	76.29	0.84
2	56.43	1200	89.15	89.50	1.07
4	96.41	1200	136.97	139.40	5.99
6	140.41	1200	204.84	204.85	0.01
3	70.42	1800	210.61	195.74	10.6
5	121.43	1500	235.38	232.92	2.16
8	181.45	1200	248.02	248.56	0.81
9	206.41	1200	259.91	261.63	3.21
7	166.42	1800	289.08	288.91	0.14

Table 3: Two metatask executions

4.2 Some Tests Performed in a Real Environment

The model is simple and the few tests performed gave good results. Some metatasks, composed of matrices multiplications, were considered. A matrix can be of size 1200, 1500 or 1800. The needs of the tasks are given in the table 2. Two executions with the error (percentage) made on the real duration of each task, are given in the table 3.

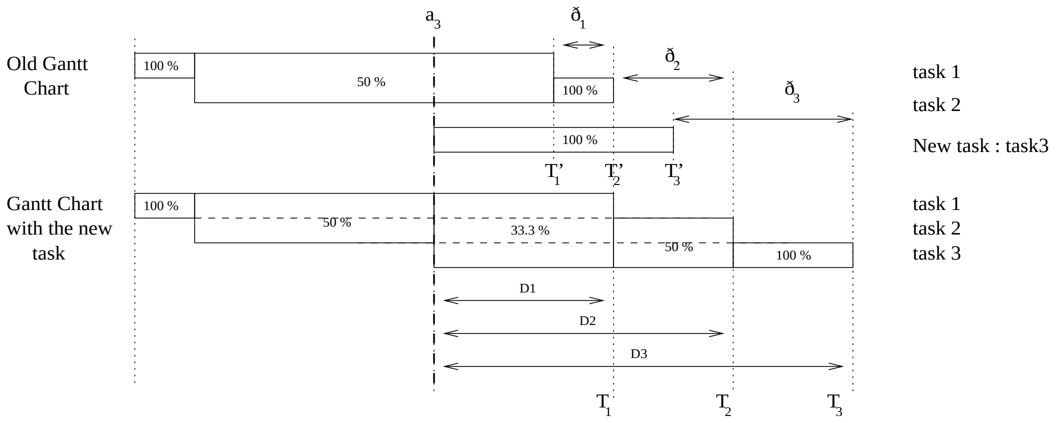


Figure 3: Notations for the historical trace use

4.3 Notations

Let a server loaded with some tasks. The oldest one, not yet finished when a new one arrives is of local number 1. If after this one $n-1$ tasks have been mapped to this server, then a new one will be of local number $n+1$. (figure 3). Note that tasks $1 < i < n+1$ can have already completed.
Let a task t mapped to the server j. Its arrival date is noted a_t and its completion date C_t. Let assume that this task is the local task i. Then we note $a_{i,j} = a_t$ the arrival date of the local task i on the server j, $d_{i,j}$ its duration on the unloaded server and $C_{i,j} = C_t$ its real completion time.

The HTM provides the following numbers after simulating the execution of tasks (fig 3) : $T'_{i,j}$ refers to the simulated finishing date of the local task i before the arrival of the new task. $T_{i,j}$ is its finishing date given after the simulation of the execution of the new task on the server j (if the model is perfect, then when the task t finishes, $C_t = T_{i,j}$). We call the remaining length of the local task i, $D_i = T_i - a_{n+1}$.
Finally, we define the *perturbation* of the new task on the local task i as $\delta_{i,j} = T_{i,j} - T'_{i,j}$.

5 Performance Metrics

The experiments explained further were conducted to compare on different metrics our new heuristics using the historical trace manager to NetSolve MCT. The main goal was to find, if possible, a heuristic that would still have good makespan performances but also give a good quality of service to each task, for they are not necessarily parts of the same application.

1. **Makespan**
   The makespan of the metatask is the completion time of the last finished task :

$$\max_{i,j}(C_{ij})$$

   Minimizing the makespan is usually the goal of the schedule of an application. The sooner the application is finished, the best it is. The makespan is dependent of the number of tasks that compose the metatask. So, for a low arrival rate, giving the gain in percentage can not give extremely good results : as it is the same metatask to be scheduled, the arrival dates are the same. It is more a gain on the flow of the last task that makes the makespan. Hence, as the number of tasks grows, the gain in percentage decreases.

2. **Sum-Flow**
   Considering the problem on the system side, it is interesting to see how much of the resources are used. The *flow-time* is the time a task t has spent in the system :

$$F_t = C_t - a_t$$

Hence, a well-suited metric for continuous job arrivals [Bak74] is the *sum-flow* :

$$\sum_t F_t$$

3. **Max-Flow** [lABCM98]
It is defined as

$$\max_t F_t$$

Thus, we can know the maximum amount of time a task has spent in the system and possibly see by this measure the impact of later arrived tasks on an already scheduled one or the choice to use a server that is slower.

4. **Max-Stretch** [lABCM98]
The *stretch* of a task t, assigned to the server j, hence being of local number i, is defined by $s_t = s_{i,j} = F_t/d_{i,j}$. We know by that factor how much a task has been slowed down relative to the time it takes on the same but unloaded server. Thus the *max-stretch* is defined as $\max_t s_t$. It is much more a metric for the client point of view. It can be qualified of *Quality Of Service* of the agent.

Note : These 2 metrics are complementary to understand what happened, however the task that has spend the most time in the system is not necessary the same than the one that has been the most delayed.

5. **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 task when comparing two heuristics. For instance, comparing the heuristics H_1 with MCT (on the *same* set of tasks $\{t_1 \ldots t_n\}$ and *same* environment), it is $\left|\{t_i | C_{t_i H_1} < C_{t_i MCT}\}\right|$

The user point of view is not that the last allocated task finishes the soonest (trying to optimize the makespan) but that his own tasks (a subset of all clients requests) finish as fast as possible. Therefore, if we can provide a heuristic where most of the tasks finish sooner than MCT's, we can claim that this heuristic, to the user point of view, outperforms MCT.

6 Mapping Heuristics

We first give the common background of the experiments done for each heuristic, and then describe the heuristics and their results (See section 4.3 for notations).

6.1 Experimental Simulation Overview

We have implemented all the mechanisms that compose NetSolve listed in section 2.3. It implies the MCT heuristic, the two load correction mechanisms when mapping a task to a server or when receiving a task completion message. It is to note that the load is boiled down to the number of task a server is running. Hence, $+100$ and -99 mechanisms are translated to $+1$ and -1 and the values used to compute the scores are far more precise than in the reality: NetSolve knows perfectly the system states. In consequence, our simulation of NetSolve MCT behaves better than the algorithm does in reality. Therefore, if we build a heuristic that outperform our simulation of NetSolve MCT, this heuristic will be likely to outperform NetSolve in reality (see section 2.3 and 4.2).

The following experiments were conducted on the same randomly generated pairs (servers,metatask), (see section 3.3), allowing to compare each proposed heuristic to the modelized NetSolve MCT. They were designed to see the quality of the schedule if the agent is more or less loaded. Hence, μ was varied from 0 to 70, where each server is executing at most one task at a given moment, for the most of the heuristics tested.

For each value of μ, we varied the number of tasks from 10 to 250. For each pair (μ,nbtasks), 1000 simulations were performed, whose results were compared to the corresponding one using the simulated NetSolve MCT. Hence, graphs are produced using the mean of the 1000 simulations : results for a heuristic represent 200000 simulations.
For three heuristics (HMCT, MP, and MSF), we have conducted experiments with a larger number of tasks, equal to 970, and means are obtained from 250 simulations. It is 48000 pairs (servers,metatask) that are analysed for each heuristic. As they confirm the previous results and tendencies, we will only refer, when studying the heuristics, to the previous one. However, the reader can find them figures 15, 16, and 17.

Results for a heuristic are given in five 3D graphics, one for each metric observed. Note that the first one concerning the number of tasks that finish sooner contains two information: the graph itself shows the percentage of tasks that finish sooner or equal to simulated NetSolve MCT ; the colored base shows the percentage of tasks that finish at the same time. Our gain is then the difference of the two numbers, and a gain is done if the value is greater than 50. For the other graphics, they show the gain in percentage using our heuristics (it is a gain when it is positive).

6.2 HMCT

HMCT is MCT that uses the historical trace management and is compared to NetSolve MCT. The historical trace manager simulates the new task on each server. Then, the scheduler selects the server that gives the best finishing date, e.g. the shortest. The algorithm is given figure 18.

Graphs 4 shows that there is a gain up to 10% on the makespan for $\mu \leq 10$. For $\mu > 30$ 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). This can be explained as follows: HMCT can map a new task to a server that is running some jobs, hence increasing their flows. For $\mu \leq 10$, the rate is so high that the flow is increased for both algorithms (hence, the gain on the Max-Flow), but the percentage of tasks that finish sooner is greater than 50%. Then, it is lower and decreases (under 20% for $\mu = 30$).

As far as the makespan is concerned, HMCT outperforms NetSolve MCT (more than 8% for $\mu \leq 20$). However, the percentage of tasks that finish sooner is poor and the Max-Stretch always negative. In a client-agent-server context, this is to improve. Moreover, MCT algorithm is not a well load-balanced heuristic : in the real case, a server can not handle too many jobs. In a high heterogeneous network, the risk is that fastest servers collapse.

6.3 Min_Max_Completion

We consider here the case of scheduling tasks that compose an only application. The last task submitted to the agent is not necessarily the one that will determine the makespan of the application. Indeed, if using MCT (or HMCT) leads to map the task for its best completion time, this is not true for the application. Hence, we propose Min_Max_Completion. The HTM computes the completion time of the task that completes the latest on each server if the new task is assigned on it. The agent chooses the server that minimizes this completion time.

Results are slightly the same as HMCT results but MMC algorithm does not give better results on the makespan than HMCT, as it was expected. Moreover, if the Max-Stretch is a little better and the Max-Flow unchanged, the number of task that finish sooner and the Sum-Flow are slightly worse.

6.4 Heuristics Relying on Minimum Perturbation

We have tested some heuristics relying on the idea to perturb the less the system. They differ in the importance given to each perturbation a new task generates on a server : each perturbation can be weighted, hence favoring for instance the finishing date of the oldest. If not said, in case of equality (for instance when the system starts), MCT is used, e.g. the chosen server is the one where the new task, the task $n+1$, finishes the soonest.

6.4.1 Min_Perturbation

From the information given by the historical trace manager, the scheduler chooses the server that minimizes $\sum_{i=1}^{n} \delta_i$ (see the algorithm figure 19).

We can see on Graphs 6 that when $nbtask > 80$, the gain on the makespan is positive (around 5% for $nbtask = 250$ and $\mu \leq 20$). MP allows to 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 of the cases, no more than one task is running on a given server). Moreover, MP allows the server to answer 90% faster in the worst case and is always better than NetSolve MCT. The Max-Flow is always better except for $\mu = 30$.

6.4.2 Min_Perturbation_load

It is the same as before, but in equal case, the heuristic considers the simulated amount of time the new task requires multiplicated by the number of tasks still running on the server. The algorithm is detailed in figure 20.

The results differ from those of MP on the Sum-Flow (worse) and on the Max-Flow (better). The gain on the Max-Flow is due to the fact that perturbation on previous tasks is minimized and less tasks are concerned by a perturbation.

6.4.3 Min_Perturbation_L_2

The perturbation is computed with the euclidian metric, hence, the server j_0 that minimizes $\sqrt{\sum_{i=1}^{n}(\delta_{i,j})^2}$ is chosen.
There is some similarities to MP, even if it gives worse results for $\mu \leq 10$ on the percentage of tasks that finish sooner and the makespan, that are less good than MCT's.

6.4.4 Min_Perturbation_masse

Four heuristics are studied. Two of them (MP_masse_decres and MP_masse_decresIncr) tend to complete the oldest running tasks, weighting their perturbations with a correponding factor, as the two other (MP_masse_cres and MP_masse_cresIncr) favors the newest. The factor is determined by the rank of the task:
In MP_masse_cres, each perturbation δ_i is weighted by i the rank of the local task i, whereas in MP_masse_cresIncr, it is by the number of oldest tasks still running (thus MP_masse_cresIncr is MP_masse_cres if tasks finish in the same order as they arrive). In MP_masse_decres, each perturbation δ_i is weighted by $n + 1 - i$, i the rank of the local task i, whereas in MP_masse_decresIncr, it is by the number of newest tasks still running (thus MP_masse_decresIncr is MP_masse_decres if tasks finish in the same order as they arrive).
The algorithms are given figure 21 and 22.

Results are equivalent for MP_masse_cres(Incr) and MP_masse_decres(Incr). Moreover, results between MP_masse_(de)cres and MP_masse_(de)cresIncr only differ on the makespan and the Sum-Flow : MP_masse_(de)cres give better results than MP_masse_(de)cresIncr.

Note : Algorithms relying on Minimum Perturbation presents a major drawback. When only one server is idle, it will be chosen by the heuristic, whatever its speed, leading to possibly bad performances. This can happen when dealing with highly heterogeneous resources and a high rate of costly requests.

6.5 Min_Length

The historical trace manager simulates the new task on each machine. The scheduler uses its information to choose the server that minimizes the quantity $\sum_{i=1}^{n+1} D_i$, that is the sum of the remaining length of each task at the new task arrival, including the new one.

For $\mu \leq 10$, the gain on the makespan is negative, then almost null as soon as $\mu \geq 20$. A gain is performed on the Sum-Flow for $\mu > 10$. Min_Length and MCT give the same results on the number of tasks that finish sooner.

6.6 Min_Sum_Flow

As described in the algorithm given figure 23, the heuristic requests the HTM to compute the whole Sum-Flow when assigning the incoming task to each server. Hence, the heuristic returns the identity of the server j_0 that minimizes the system sum flow, e.g. $\min_j(\sum_{k \neq j, i=1}^{i=n}(T'_{i,k} - a_{i,k}) + \sum_{i=1}^{i=n+1}(T_{i,j} - a_{i,j}))$. But as the difference between two values is only due to perturbations and to the new simulated task duration, the HTM only needs to compute $\sum_{i=1}^{n} \delta_{i,j} + T_{n+1,j} - a_{n+1,j}$ for each server j, that is the perturbation of the last task on the server plus the manager estimated length of the new task.
MSF achieves to gain on the makespan like HMCT does, e.g. around 10% for $\mu \leq 10$. The gain on the Max-Stretch is positive for $\mu \leq 30$, the gain Max-Flow is always positive with a peak to 23% for $\mu = 30$. Surprisingly, the gain on the Sum-Flow is lower than expected. The number of task that finish sooner is lower than MCT's.

Since the cost of the new task is computed, the algorithm begins like HMCT. Thus, the mapping decision can be viewed as partly composed of HMCT and MP part. Moreover, MSF results can be considered (except on the Max_Flow) like a mean between MP and HMCT.

7 Conclusion and Future Work

In this paper we have studied the problem of scheduling a metatask on a set of servers in the "client-agent-server" model.
We have introduced the concept of *historical trace* in order to better predict the behavior of each server as well as the impact of the mapping of the tasks onto the environment. We have proposed new heuristics based on the *historical trace* concept. Among them,

three heuristics present good results: HMCT is a variant of MCT as used in NetSolve ; Min_Perturbation tends to minimize the impact on already mapped tasks ; MSF tends to gather the advantages of the two previous heuristics.
Our simulation experiments show that, for a negligible cost: HMCT performs better than MCT on the makespan, but does not give the same quality of service than other does for same makespan results ; Min_Perturbation and Min_Sum_Flow outperform on most cases for most of the metrics observed, like the makespan, our simulation of NetSolve scheduling heuristic (MCT) which has a far better knowledge of the environment than in reality.
However, in degenerated cases, Min_Perturbation has some drawbacks that MSF does not have. Therefore the MSF heuristic is a good candidate for scheduling metatasks in the general case.

Our future works are directed towards implementing these heuristics into the NetSolve code in order to perform tests on real applications and real environments. Next, we plan to build a scalable version of the historical trace manager 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+98] Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau Bölöni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys et Bin Yao, *A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems*, Proceedings of the 17th IEEE Symposium on Reliable Distributed System, 1998.

[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.

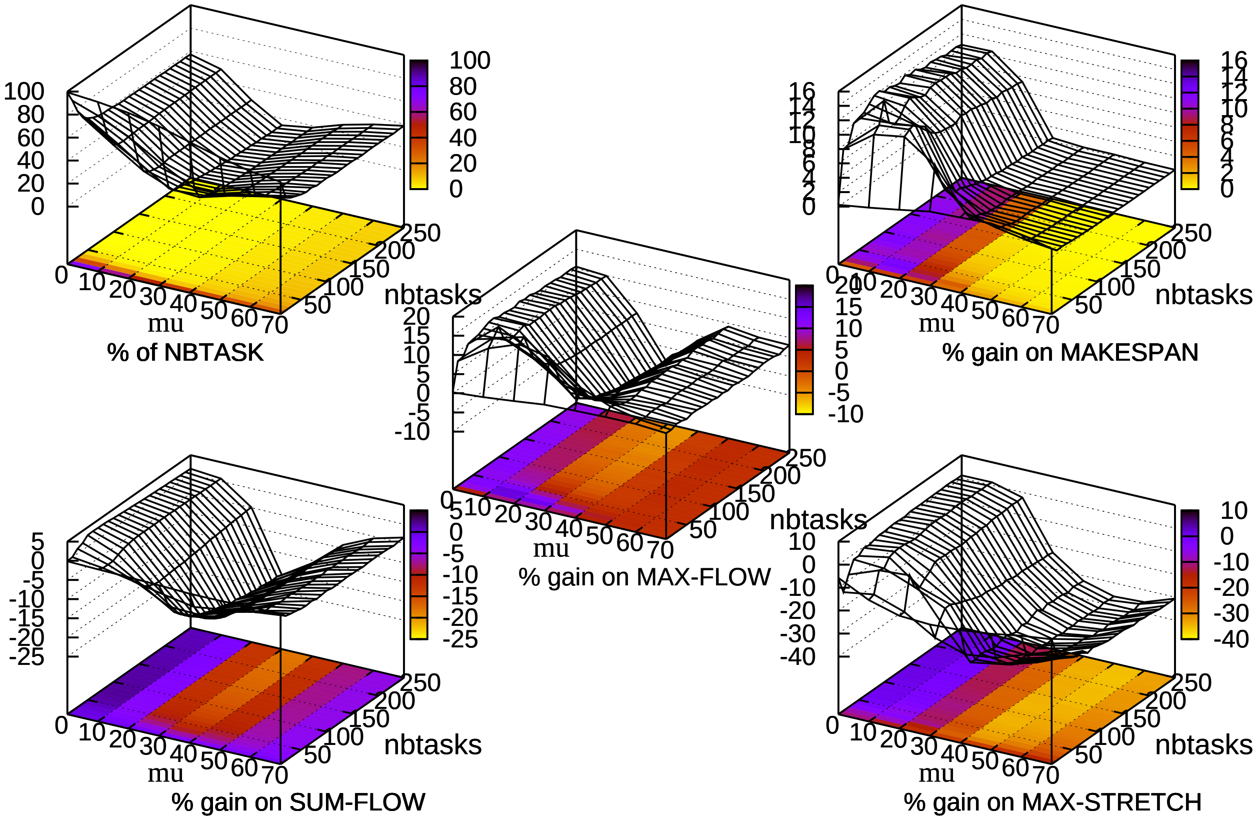


Figure 4: Results for HMCT vs MCT on 25 servers, 250 tasks

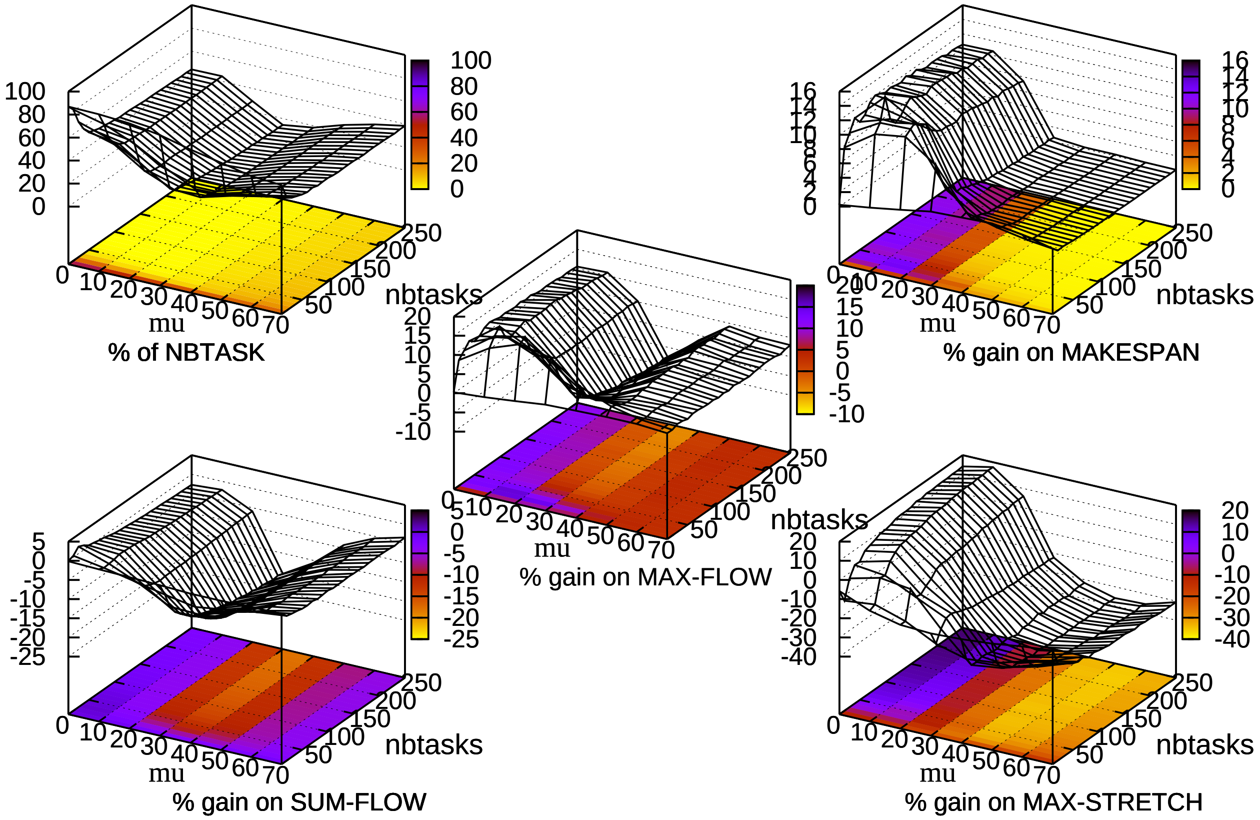


Figure 5: Results for Min_Max_completion vs MCT on 25 servers, 250 tasks

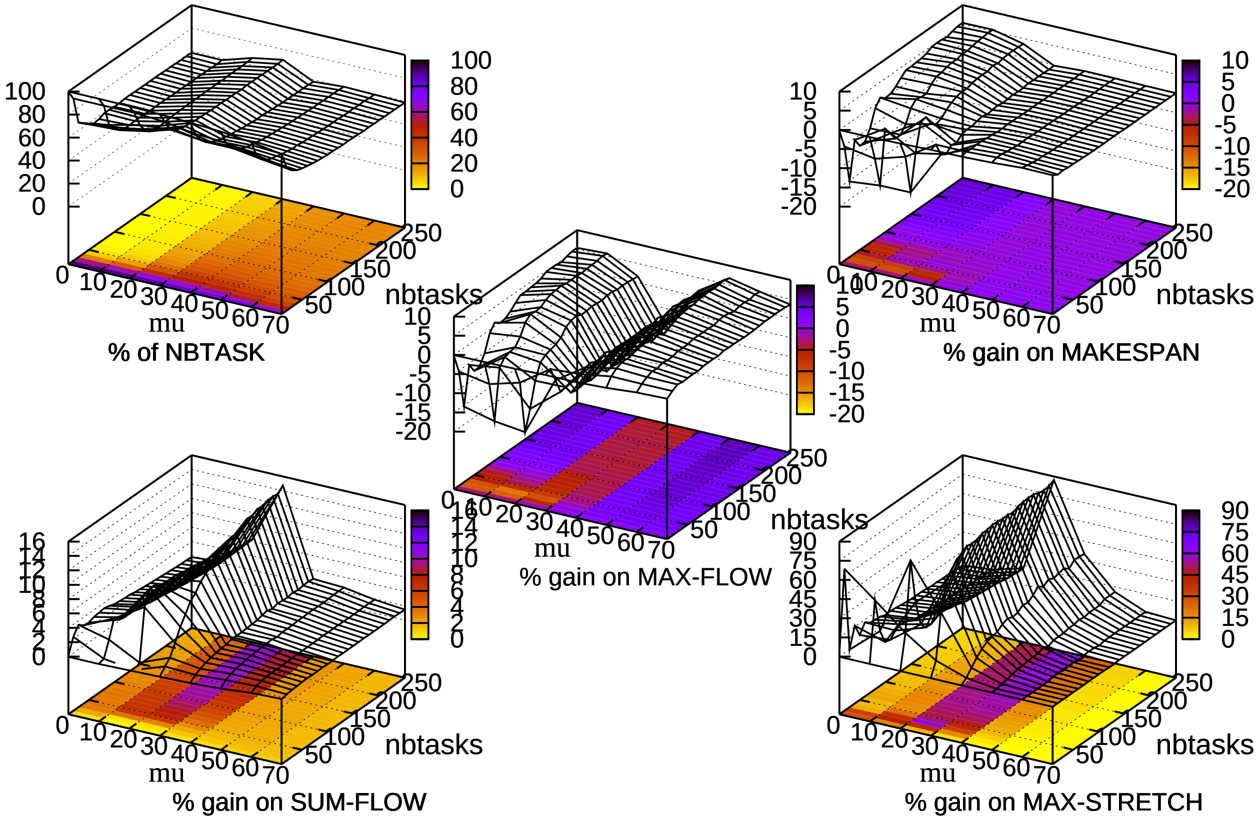


Figure 6: Results for Min_Perturbation vs MCT on 25 servers, 250 tasks

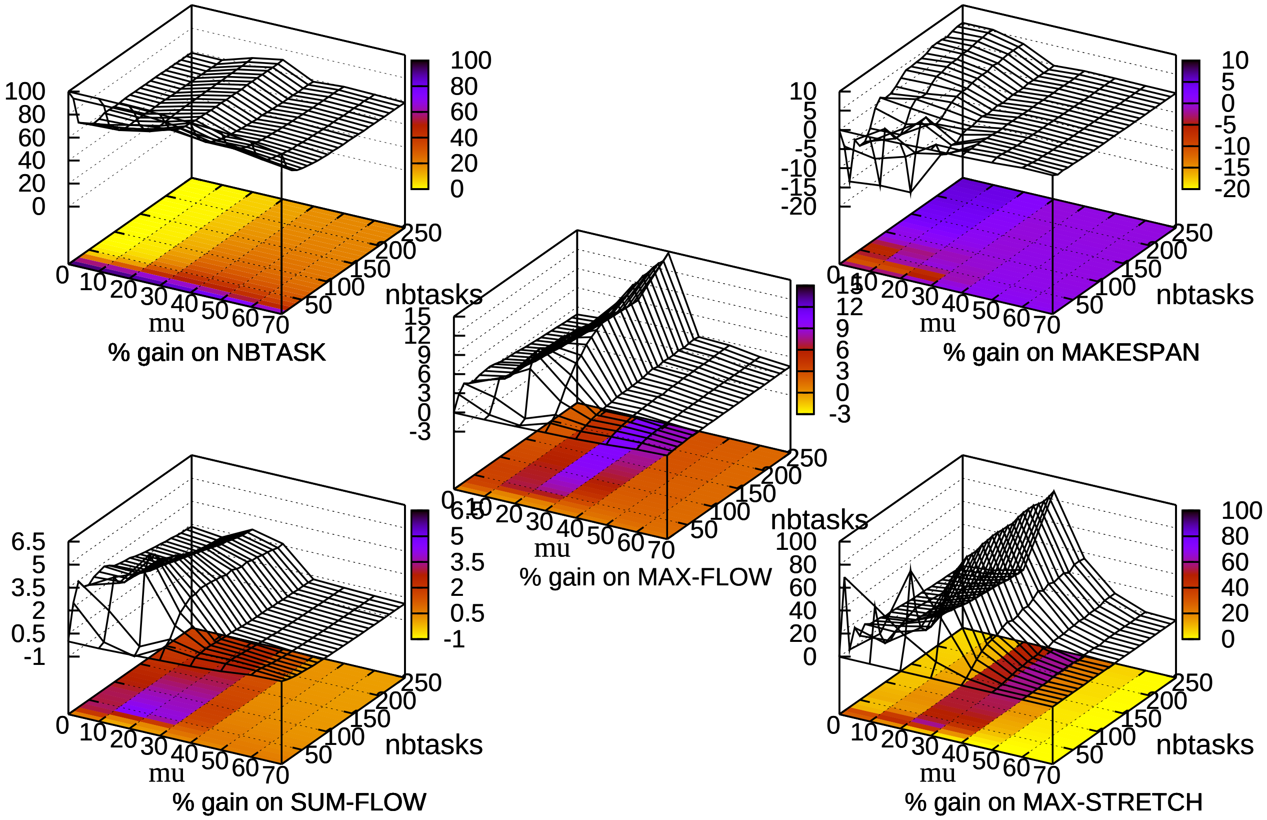
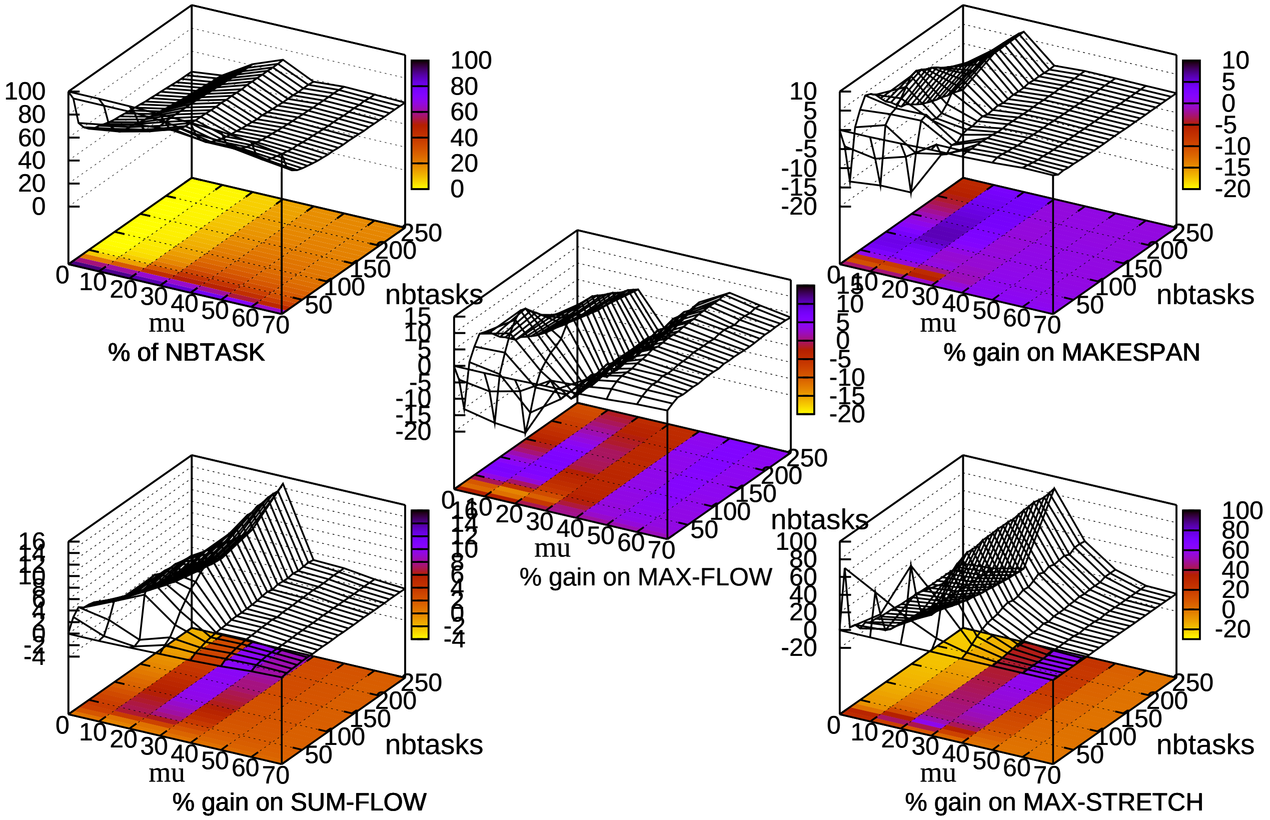


Figure 7: Results for Min_Perturbation_load vs MCT on 25 servers, 250 tasks



% of NBTASK

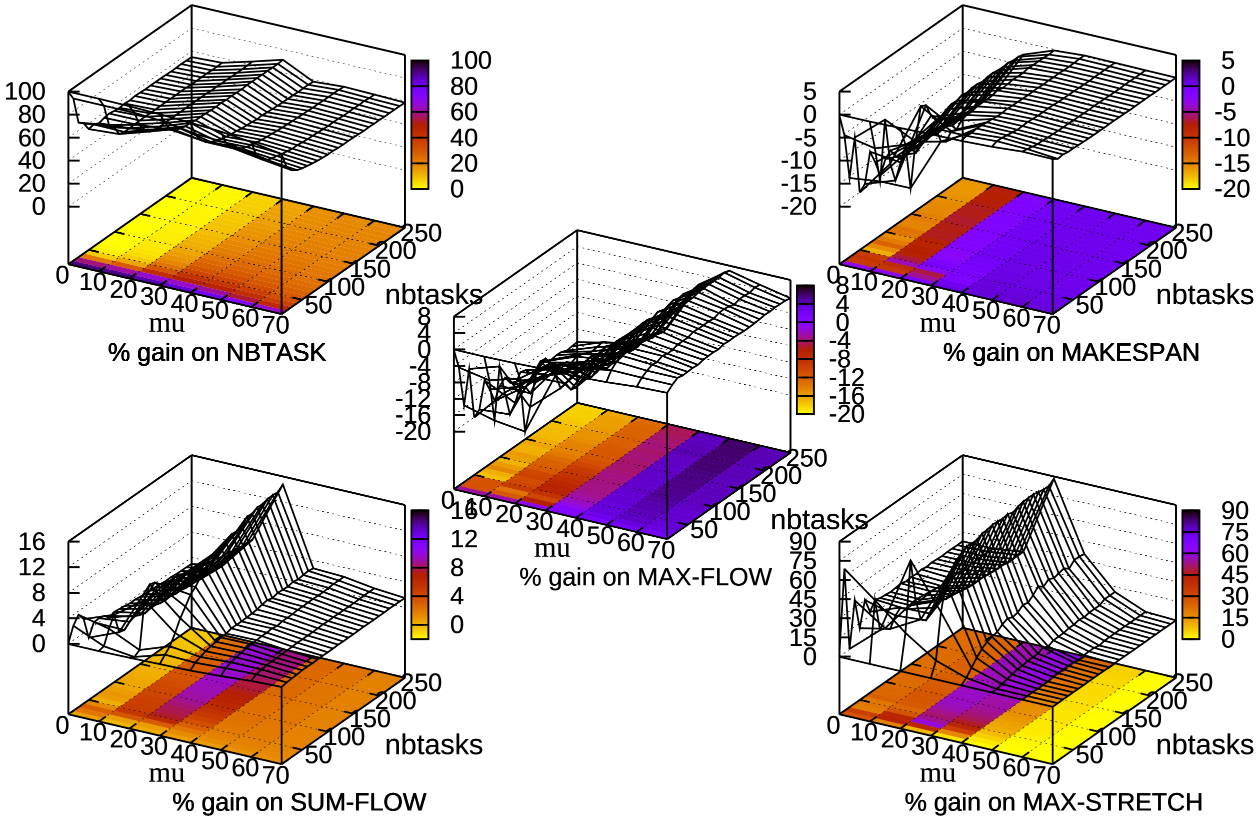
% gain on MAKESPAN

% gain on MAX-FLOW

% gain on SUM-FLOW

% gain on MAX-STRETCH

Figure 8: Results for Min_Perturbation_L_2 vs MCT on 25 servers, 250 tasks



% gain on NBTASK

% gain on MAKESPAN

% gain on MAX-FLOW

% gain on SUM-FLOW

% gain on MAX-STRETCH

Figure 9: Results for Min_Perturbation_mass_cres vs MCT on 25 servers, 250 tasks

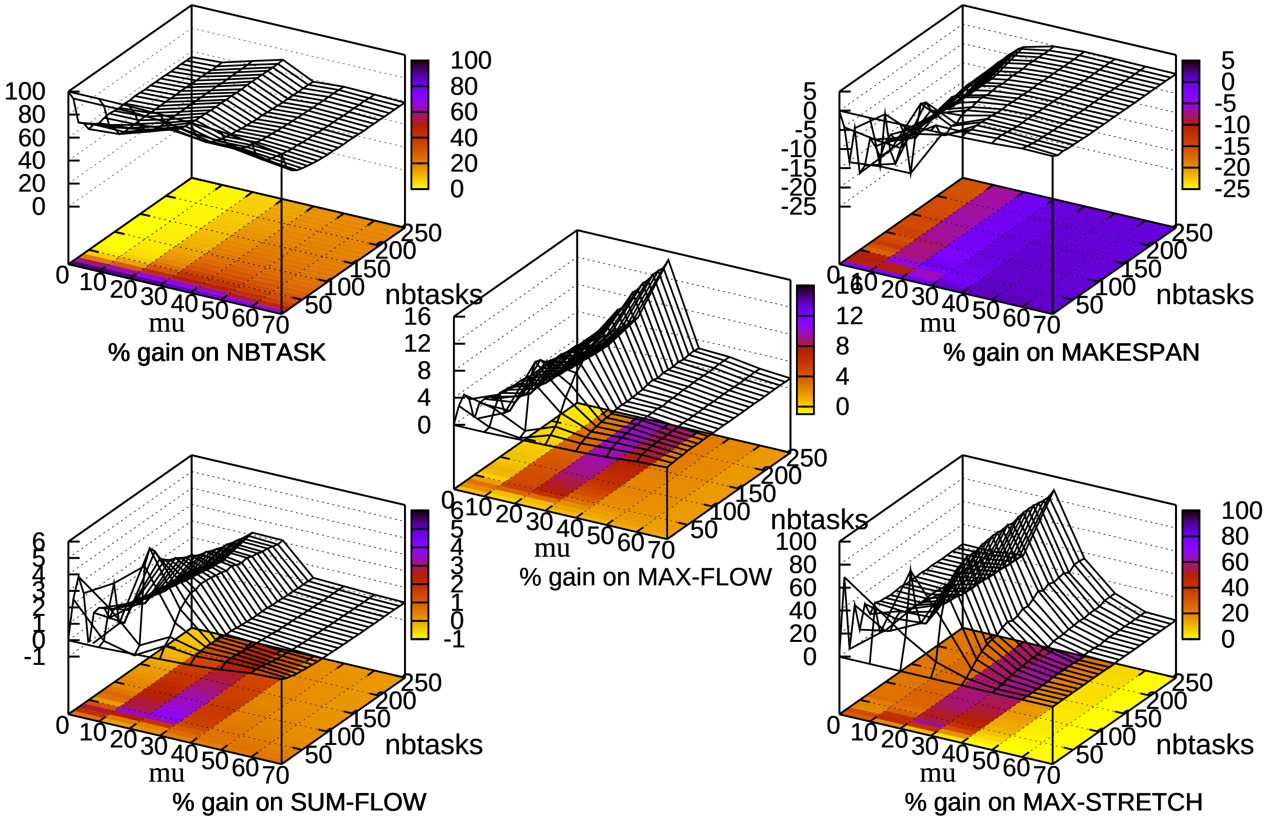


Figure 10: Results for Min_Perturbation_mass_cresIncr vs MCT on 25 servers, 250 tasks

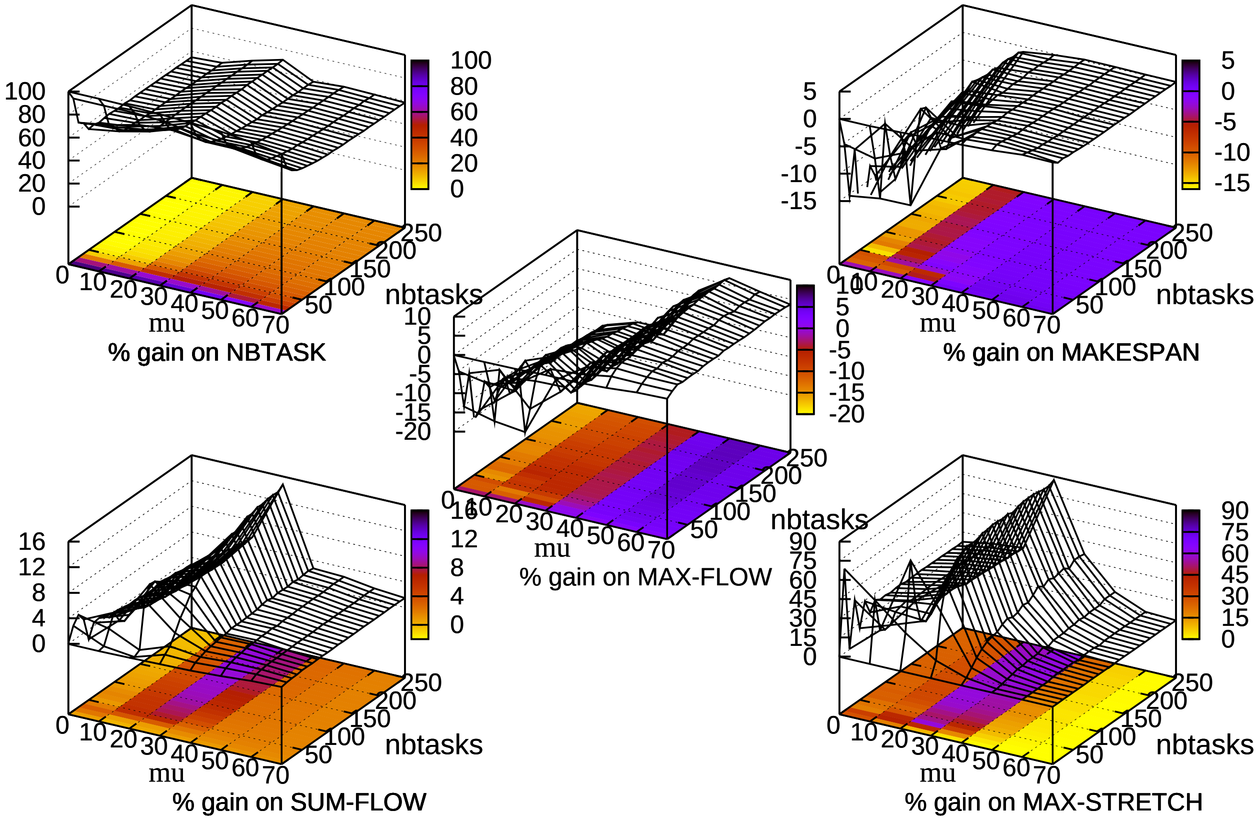


Figure 11: Results for Min_Perturbation_mass_decres vs MCT on 25 servers, 250 tasks

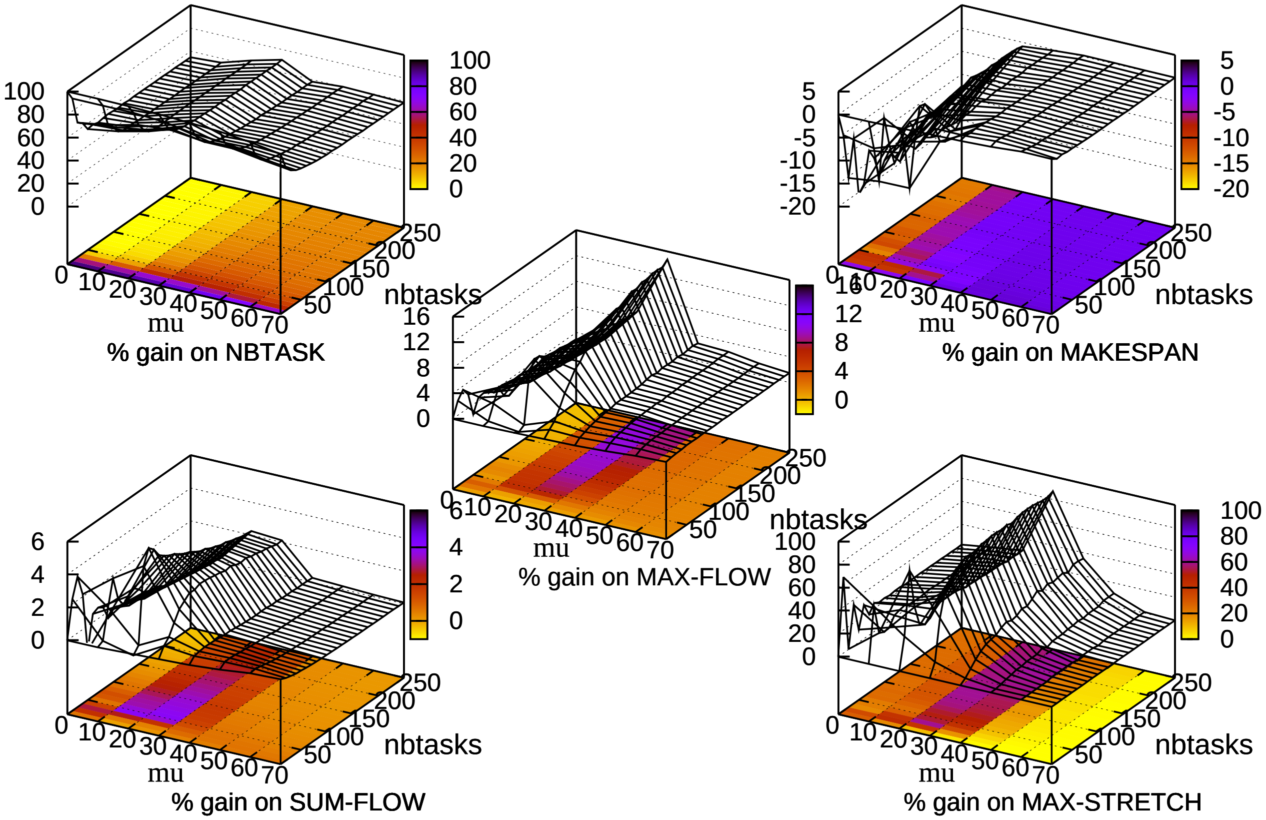
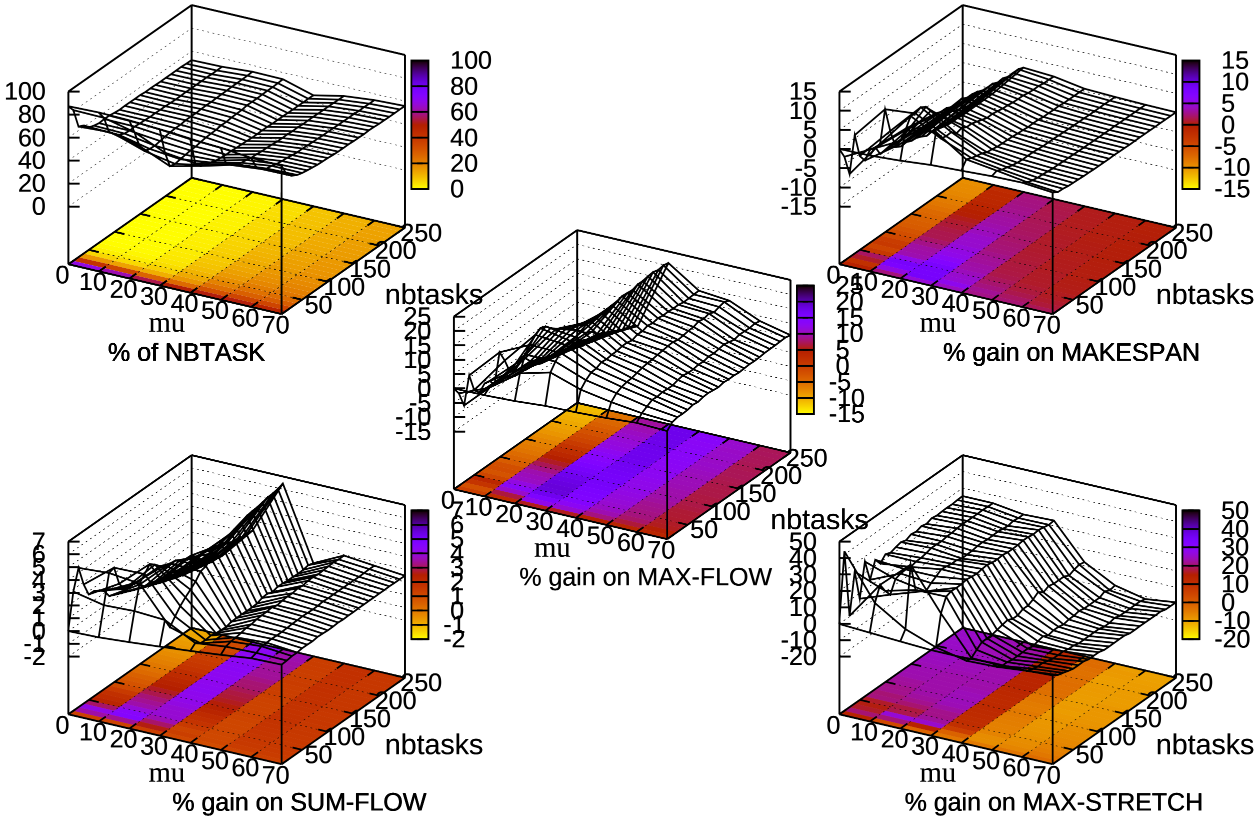


Figure 12: Results for Min_Perturbation_mass_decresIncr vs MCT on 25 servers, 250 tasks


% of NBTASK

% gain on MAKESPAN

% gain on MAX-FLOW

% gain on SUM-FLOW

% gain on MAX-STRETCH

Figure 13: Results for Min_length vs MCT on 25 servers, 250 tasks

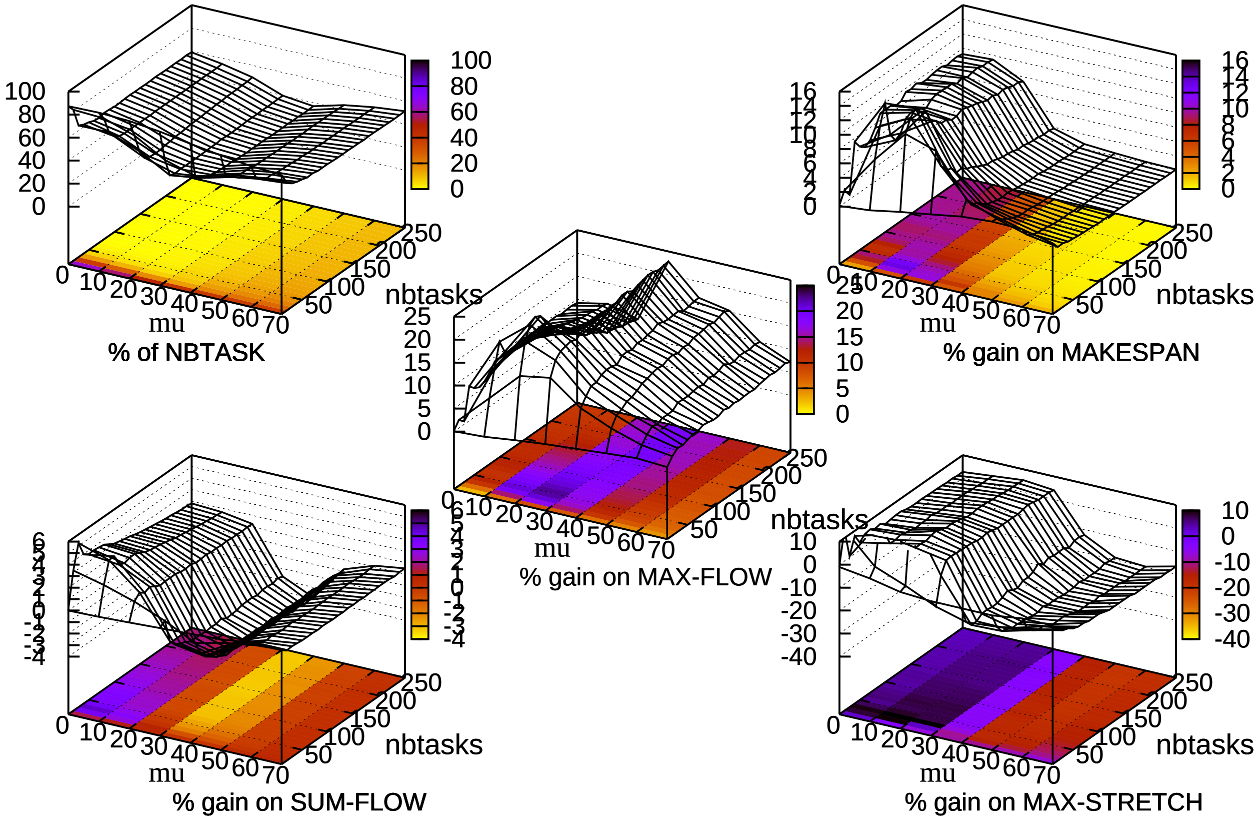
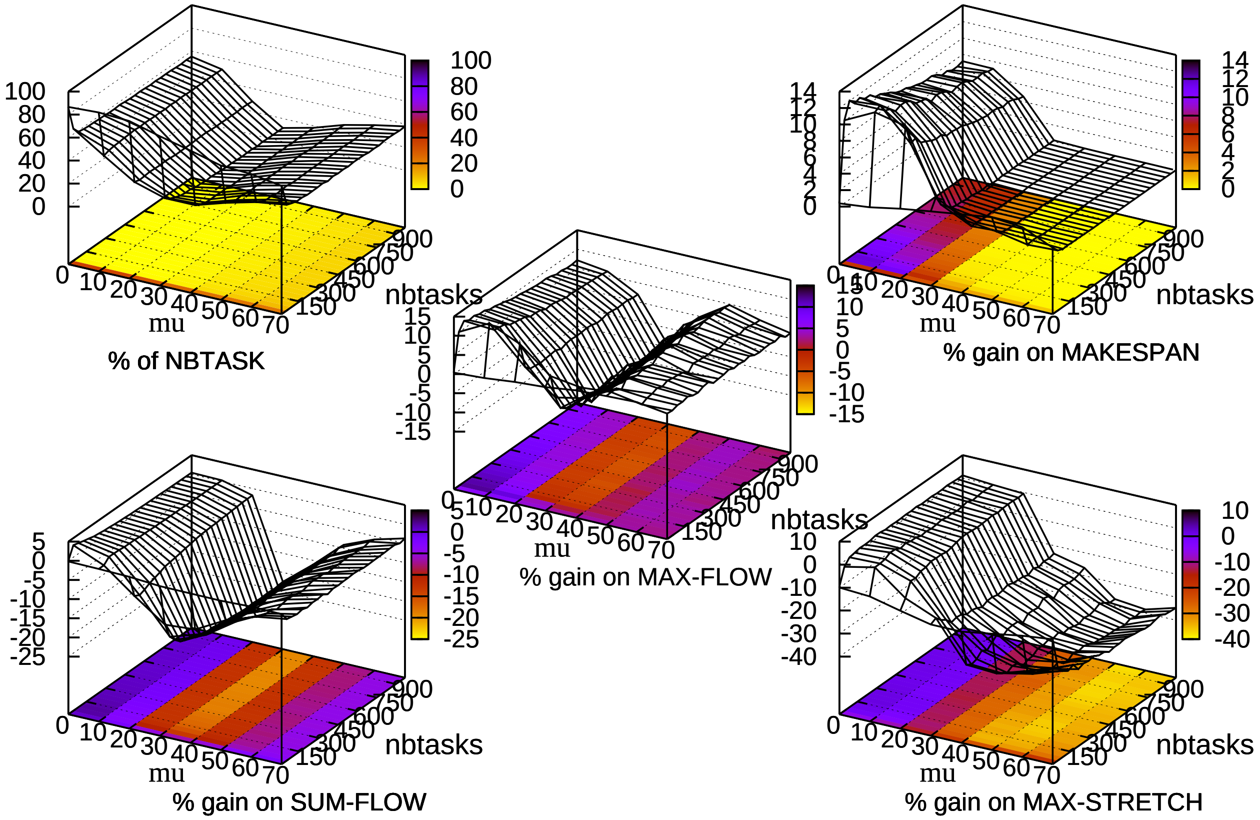


Figure 14: Results for MSF vs MCT on 25 servers, 250 tasks


% of NBTASK

% gain on MAKESPAN

% gain on MAX-FLOW

% gain on SUM-FLOW

% gain on MAX-STRETCH

Figure 15: Results for HMCT vs MCT on 25 servers, for 970 tasks

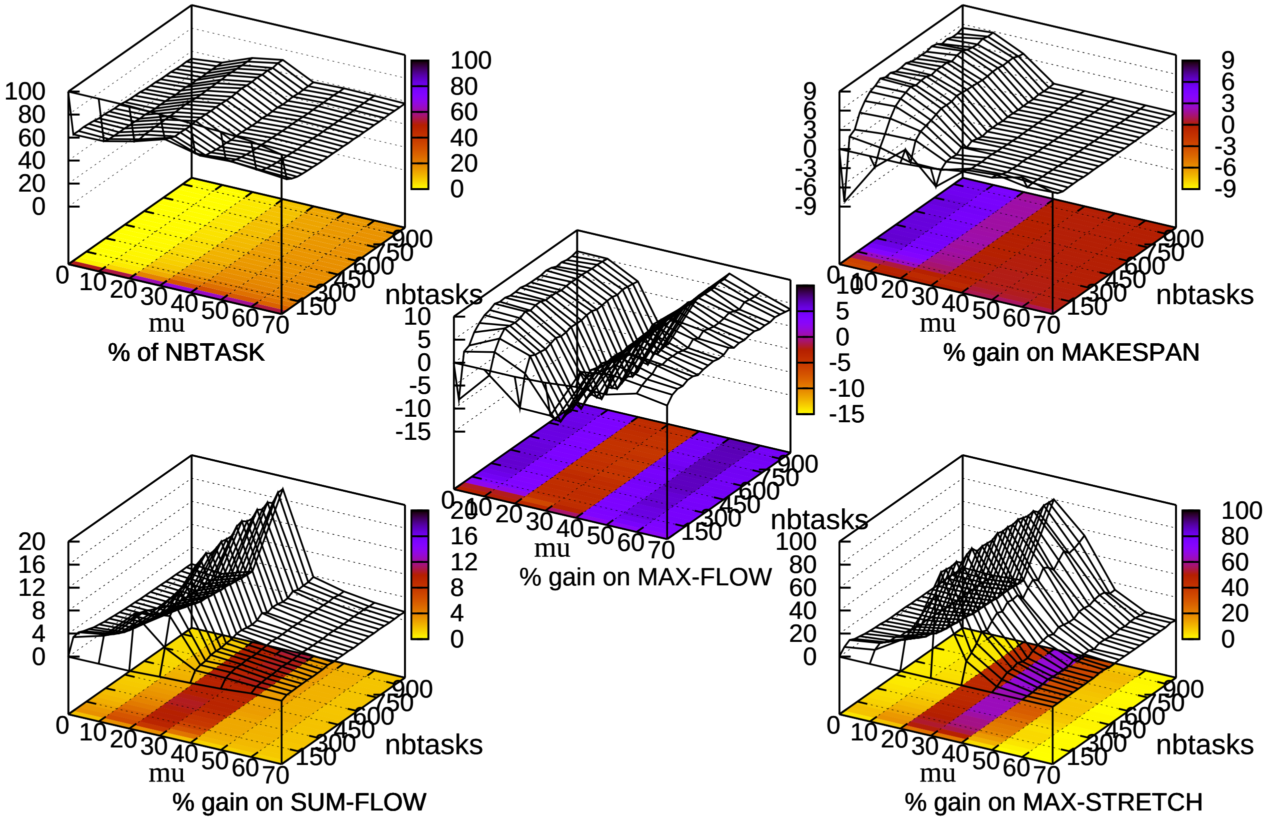
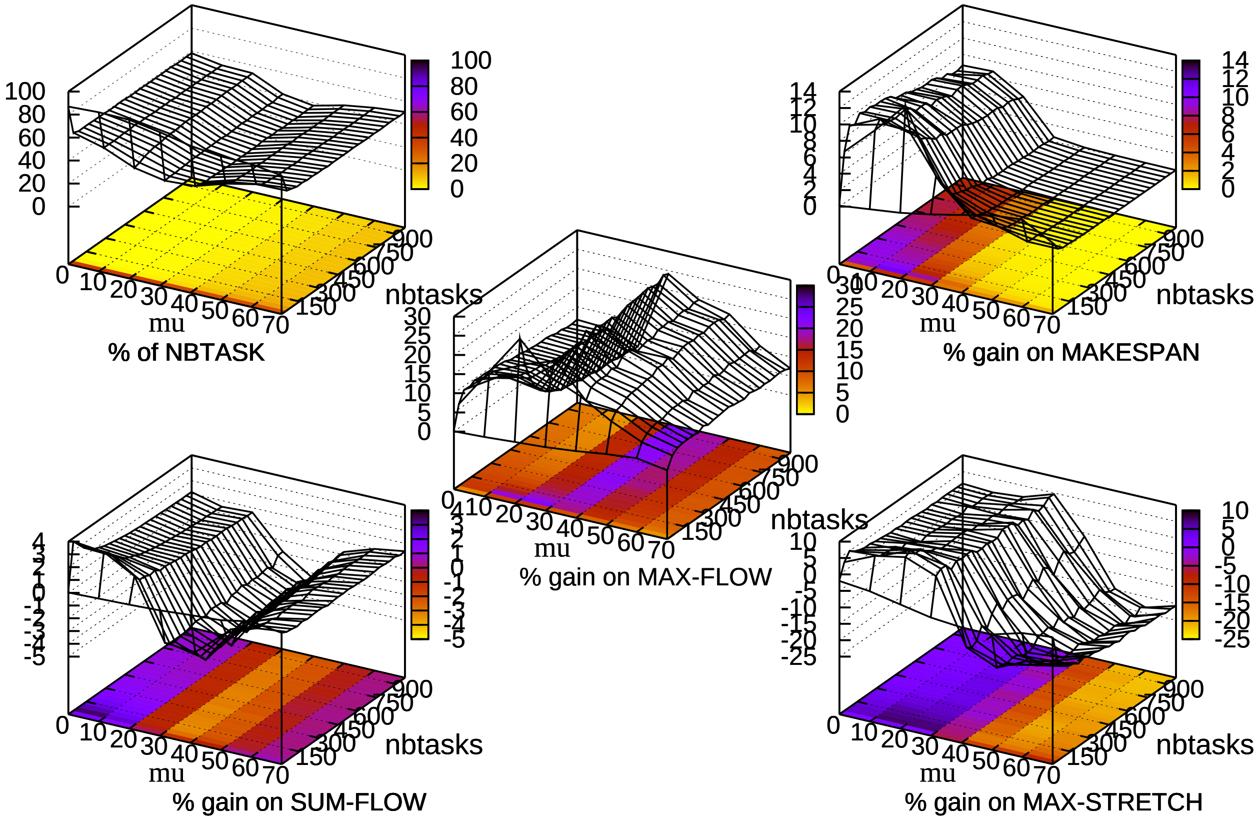


Figure 16: Results for MP vs MCT on 25 servers, for 970 tasks



% of NBTASK

% gain on MAX-FLOW

% gain on MAKESPAN

% gain on SUM-FLOW

% gain on MAX-STRETCH

Figure 17: Results for MSF vs MCT on 25 servers, for 970 tasks

[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.

[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.

[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.

[WSH99] R. Wolski, N.T. Spring et J. Hayes, *The network service : A distributed resource performance forecasting service for metacomputing*, Journal of Future Generation Computing Systems **15** (1999), no. 5-6, 757–768.

```
For each new task $t$
    For each server $j$ that can resolve the new submitted problem
        The task $t$ is of local number $l_j$
        Ask the HTM to compute $C_{l_j,j}$
    Map task $t$ to the server $j_0$ such that $C_{l_{j_0},j_0} = \min_j C_{l_j,j}$
    Tell the HTM that the task $t$ is allocated to the server $j_0$
```

Figure 18: *HMCT algorithm*

```
For each new task $t$
    For each server $j$ that can resolve the new submitted problem
        Ask the HTM to compute $P_j = \sum_i \delta_{i,j}$
    If all $P_j$ are equal
        map task to the server $j_0$ that minimizes $C_{n+1,j}$
    Else Map task $t$ to the server $j_0$ such that $P_{j_0} = \min_j P_j$
    Tell the HTM that the task $t$ is allocated to the server $j_0$
```

Figure 19: *MP algorithm*

```
For each new task $t$
    For each server $j$ that can resolve the new submitted problem
        Ask the HTM to compute $P_j = \sum_i \delta_{i,j}$
        Let $NB_j$ be the number of tasks still running
    If all $P_j$ are equal
        map task to the server $j_0$ that minimizes $NB_j * D_{n+1,j}$
    Else Map task $t$ to the server $j_0$ such that $P_{j_0} = \min_j P_j$
    Tell the HTM that the task $t$ is allocated to the server $j_0$
```

Figure 20: *MP_load algorithm*

```
For each new task $t$
    For each server $j$ that can resolve the new submitted problem
        Ask the HTM to compute $P_j = \sum_i i * \delta_{i,j}$
    If all $P_j$ are equal
        map task to the server $j_0$ that minimizes $C_{n+1,j}$
    Else Map task $t$ to the server $j_0$ such that $P_{j_0} = \min_j P_j$
    Tell the HTM that the task $t$ is allocated to the server $j_0$
```

Figure 21: *MP_masse_cres algorithm*

```
For each new task $t$
    For each server $j$ that can resolve the new submitted problem
        $NB = 1$
        For each task $i$ still running, sorted in their arrival date
            Ask the HTM to compute $P_j + = NB * \delta_{i,j}$
            $NB + = 1$
    If all $P_j$ are equal
        map task to the server $j_0$ that minimizes $C_{n+1,j}$
    Else Map task $t$ to the server $j_0$ such that $P_{j_0} = \min_j P_j$
    Tell the HTM that the task $t$ is allocated to the server $j_0$
```

Figure 22: *MP_masse_cresIncr algorithm*

```
For each new task $t$
    For each server $j$ that can resolve the new submitted problem
        The task $t$ is of local number $l_j$
        Ask the HTM to compute $P_j = \sum_i \delta_{i,j} + T_{l_j,j} - a_{l_j,j}$
    Map task $t$ to the server $j_0$ such that $P_{j_0} = \min_j P_j$
    Tell the HTM that the task $t$ is allocated to the server $j_0$
```

Figure 23: *MSF algorithm*

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399