Resource Re-allocation for Data Inter-dependent Continuous Tasks in Grids

Valeriia Haberland[1], Simon Miles[2], and Michael Luck[2]

[1] Tungsten Centre for Intelligent Data Analytics
Goldsmiths, University of London
v.haberland@gold.ac.uk
[2] Department of Informatics, King's College London
simon.miles, michael.luck@kcl.ac.uk

Abstract. Many researchers focus on resource intensive tasks which have to be run continuously over long periods. A Grid may offer resources for these tasks, but they are contested by multiple client agents. Hence, a Grid might be unwilling to allocate its resources for long terms, leading to tasks' interruptions. This issue becomes more substantial when tasks are data inter-dependent, where one interrupted task may cause an interruption of a bundle of other tasks. In this paper, we discuss a new resource re-allocation strategy for a client, in which resources are re-allocated between the client tasks in order to avoid prolonged interruptions. Those re-allocations are decided by a client agent, but they should be agreed with a Grid and can be performed only by a Grid. Our strategy has been tested within different Grid environments and noticeably improves client utilities in almost all cases.

Keywords: Continuous inter-dependent tasks, Resource re-allocation, Client's decision-making mechanism

1 Introduction

Recently much research has focused on smart systems which, for example, monitor the level of pollution in the environment [5]. These systems have to acquire and process data continuously to be able to produce up-to-date results, and the tasks which process these data have to be run continuously and for long periods of time [7, 8]. It is desirable for these tasks to run without interruption, but short interruptions whose duration depends on the nature of a task, may not affect significantly the controlled parameters e.g., temperature. These tasks also have to be executed for so long periods of time that a Grid is unable or unwilling to allocate them for the whole period at once. This means that a task will be interrupted after some agreed period of time and it has to obtain new resources [8]. A task might also be interrupted unexpectedly due to a resource failure. Here, we assume that the resource availability changes near-periodically over time [1, 9, 11], allowing its peaks to be approximately overseen in the future [8].

These tasks can also depend on each others' data, i.e. one task might require data from other tasks in order to run. For example, two data streams which

This is the accepted manuscript version published in LNCS by Springer, 2017.
The final version can be found at: https://doi.org/10.1007/978-3-319-59294-7_16

monitor temperature and humidity are linked as the weather observation in the airport [12]. Other use cases include traffic monitoring (control) on a road [19], where the vehicles' speed and location are streamed in order to identify traffic congestion. These scenarios show continuous tasks linked in terms of data and it would be realistic for these tasks to run simultaneously.

Some research e.g., [10, 13], focuses on execution of inter-dependent tasks, but it is generally lacking decision making mechanisms for a client in respect of allocation and execution of such tasks. The dependence among tasks is often depicted in terms of the data exchange and has an explicit connection between sender and recipient tasks. In our model, the tasks do not have just an explicit dependence, but also an implicit one which means that a failure of the recipient-task affects the execution of the corresponding sender-task as much as the sender-tasks affect the execution of their recipients. We also take into account that tasks are not executed just once, but have to be executed continually over the long term, where any failed tasks might affect a controlled parameter.

Hence, if data is not received in time by a recipient-task, then the client's system will produce erroneous results to some degree, i.e. the longer this delay, the larger the probability that the last received data from a sender-task is significantly different from the current data that would be received. As time passes, the recipient-task has to stop eventually, avoiding to produce substantially deviated results. In comparison, other work generally does not focus on how a client agent can avoid or shorten these delays in the case of highly contested Grid resources and how those delays may affect its system.

We propose a new resource re-allocation strategy, *SimTask*, for a client which allows a client agent to exchange the allocated resources among its own tasks by negotiating with the *Grid Resource Allocator* (GRA). The aim of this exchange is to avoid long interruptions which cause a significant change in the parameter controlled by the interrupted task e.g., a significant drop in temperature. This strategy incorporates a decision-making mechanism for a client agent (referred further as a client) to initiate resource re-allocation and choose the most appropriate task for donating resources. The agents' abilities [21] as to decide autonomously and respond actively to any changes are crucial for this mechanism, considering the large number of negotiating agents at the same time.

The paper is structured as follows.[1] Section 2 discusses related work in respect of the inter-dependent tasks. Then, Section 3 describes the formal model, while Section 4 presents our SimTask re-allocation strategy. The evaluation results are discussed in Section 5, and Section 6 concludes the paper.

2 Related Work

In this paper, we focus on the tasks which run near-continuously [7, 8] over time, and also depend on each other's data. In other words, each task sends a data point to another task, while processing new input data. Hence, these tasks have

[1] The authors thank King's College London for sponsoring this work as a part of Ph.D research [6].

to be run simultaneously in terms of processing each input data point as soon as it has arrived, and repeat this processing over time. Much research e.g., [10, 13, 15, 23], considers processing of inter-dependent tasks in Grid systems where dependencies are presented as a *directed acyclic graph*. In particular, Meriem and Belabbas [15] dynamically allocate tasks to resources, which arrive as a continuous stream over time. Dynamic allocation is meant to respond to any resource availability changes in a Grid, and resolve the problem of load-balancing at runtime. Although all this work considers task dependencies in resource allocation, tasks are not considered to be repeated continuously in real-time. Nevertheless, this research describes relevant concepts which can be applicable to continuous tasks such as *spare time* [23], which defines the maximal time of task execution before it affects the schedule of dependent tasks. Other work considers a *cyclic task graph* [17, 18, 22], where tasks are executed repeatedly over time and each task in a cycle obtains and sends data. Here, the cycles of task dependencies are represented in terms of data, instructions, etc.

There are other examples of inter-dependent tasks that were discussed in the literature. For example, the work [14] focuses on accomplishing a high-level task by completing a number of *time-constrained possibly inter-dependent other tasks* e.g., gathering information from the Web in order to offer appropriate products to the customers. Motwani et al. [16] focus on *continuous queries* (e.g. continuous tasks), which process stream data from multiple sources. A continuous query is the type of query which is issued once for a particular data-type and then runs continuously, updating a client with new results without being issued repeatedly [3, 20]. In the work [16], one query may consist of a number of sub-queries (operators), where the outputs of these sub-queries might be shared with other queries or sub-queries. Although this work discusses the techniques to approximate the query outputs in the case of scarce resources, it does not focus on how data delays or failed sub-queries might affect the results from other sub-query or query, or whether the latter query can even be performed.

Different platforms (engines) e.g., *Apache Storm* [2], attempt to solve the problems of scalability, performance and memory usage in terms of execution of data streams. However, the problem of tasks' inter-dependencies as discussed above and how they can be run without some input data is not the focus of these engines. Note that in an open and dynamic computational environment such as a Grid, where other clients also require resources, it might be difficult to re-allocate a task without affecting other clients' interests.

3 Formal Model

In this work, we consider that tasks have inter-dependencies, where some tasks, *sender-tasks*, send data to other tasks, *recipient-tasks*. The dependencies among tasks are presented as a *rooted tree* Tr, where data streams flow from the bottom to the top of this tree (i.e. from leaf to root). Each node of this tree denotes a task i and each edge indicates a data inter-dependence between sender-task $i \in \mathbb{N}$ and recipient-task $j \in \mathbb{N}$ with a weight $\alpha_{i,j} \in [0, 1]$. The weight denotes the *level*

of importance of the data from a particular sender-task for the corresponding recipient-task, i.e. each edge has a direction from the lower-layer tasks (senders) towards the upper-layer tasks (recipients) of the tree. In this way, some tasks in the middle of the tree are also senders and recipients at the same time. We assume that each sender-task has only one corresponding recipient-task, but every recipient-task might have one or more sender-task(s). We also assume that the sum of weights for all sender-tasks which are connected directly to the same recipient-task is equal to 1.0, and the smaller this weight $\alpha_{i,j}$, the less impact sender-task, i, has on the work of recipient-task, j. This model of task inter-dependencies can follow, for example, a scenario of data aggregation from multiple sources by counting, adding, etc. the data instances over some time [4]. We also consider a *sub-tree* $sTr_k \in Tr$ with the root task $k \in \mathbb{N}$.

Here, we define an abstract *parameter* P which is estimated by client tasks. This parameter $P_{i,S_i}(t) \in \mathbb{R}$ for task $i \in \mathbb{N}$ with the corresponding set of direct sender-tasks $S_i = \{m, ..., k\}$ at time t is a real-life characteristic (e.g. temperature), which is continuous or can be presented as continuous over time $t \in \mathbb{R}$, considering $|(P_{i,S_i}(t + \Delta t) - P_{i,S_i}(t))/P_{i,S_i}(t)| \ll 1$ where $\Delta t \in \mathbb{R}$ is an arbitrary small time step. The parameter $P_{i,S_i}(t)$ is estimated directly by task i, if this task belongs to the lowest layer of a tree, i.e. $S_i \in \emptyset$. If task i belongs to any upper layer of a tree, i.e. $S_i \neq \emptyset$, then $P_{i,S_i}(t)$ is estimated as a linear combination of all parameters $P_{j \neq i,S_j}(t)$ sent by its sender-task(s)[2] $P_{i,S_i}(t) = \sum_{j \in S_i} \alpha_{j,i} \times P_{j \neq i,S_j}(t)\,,\ S_i \neq \emptyset,\ i, j \in \mathbb{N}$, where S_j can be empty set.

3.1 Status and Layer

In our model, each task i has its *status* $Status_i(t)$ of execution at time t, which can be: *'interrupted'* means a task is not running and does not possess resources; *'stopped'* denotes a task is not running (i.e. its recipient-task is interrupted or it has produced inaccurate results for too long), but it possesses resources; *'inaccurate'* means a task is running, but at least one of its sender-task(s) either does not send any data or sends inaccurate data; *'accurate'* means a task is running and all its sender-tasks send accurate data. A task produces inaccurate data when at least one of its sender-tasks has status other than 'accurate'.

We also consider that each task i belongs to a specific layer $Layer_i$ in a tree, and sends its data, except the root task, to the corresponding recipient-task j which belongs to the nearest upper layer, i.e. $Layer_j = Layer_i + 1$. Note that the lower layer tasks are stopped when a recipient-task on the top of a sub-tree or the whole tree is stopped or interrupted. This means that if sender-tasks have no tasks to send their data to, they are stopped. This dependence shows the continuous and real time nature of the tasks.

3.2 Damping and Delay Time

As we discussed a notion of short interruption, we define a *damping time* which determines for how long a task can be interrupted or stopped without substantial

[2] A linear combination is chosen for a greater clarity of evaluation of SimTask.

negative consequences for parameter estimation e.g., a significant rise in temperature. If any task has been interrupted or stopped, then it does not estimate this parameter any more. In this way, a parameter changes in a way which is not under control of a client. Hence, the longer this task is not running, the higher probability that the change of this parameter might have a substantial negative effect for a client. We consider that this effect occurs when the damping time $\tau_i^{dam}(t_d)$, starting at time $t_d \in \mathbb{R}$, has passed for task i. That is, the absolute difference $\Delta P_{i,S_i}(t)$ between the last produced value of parameter $P_{i,S_i}(t_d)$ by task i before interruption and the linearly extrapolated value of this parameter $P_{i,S_i}^{ex}(t)$ at time t becomes larger than the predefined threshold $\eta_i^{dam} \in \mathbb{R}$. This threshold is determined by the nature of this parameter.

A *delay time* determines for how long a task can be running when it has to use inaccurate data for its calculations and it stops after this time. A delay time $\tau_i^{del}(t_{dl}, t)$, starting at time $t_{dl} \in \mathbb{R}$, for recipient-task $i \in \mathbb{N}$ is the duration of time when this task can still run, but it has to use inaccurate input data for its calculations due to the interruption of some (at least one) sender-task(s) from its sub-tree sTr_i. This time ends when the absolute difference $\Delta P_{i,S_i}(t) = \sum_{j \in S_i} \alpha_{j,i} \Delta P_{j \neq i, S_j}(t)$, $S_i \neq \emptyset$ at time t becomes larger than the predefined threshold $\eta_i^{del} \in \mathbb{R}$, where $\eta_i^{del} < \eta_i^{dam}$. Note that the difference $\Delta P_{i,S_i}(t)$ is a linear combination of such differences for the lower layer tasks which belong to a sub-tree sTr_i and have the statuses of execution as 'inaccurate', 'stopped' or 'interrupted'. As for the lowest layer tasks, i.e. $S_j = \emptyset$, these differences at time t are calculated as $\Delta P_{j,S_j}(t) = P_{j,S_j}(t_{dl}) - P_{j,S_j}^{ex}(t)$, where $P_{j,S_j}(t_{dl})$ is the last value of parameter produced by task j before its interruption at time t_{dl}. The difference $\Delta P_{i,S_i}(t)$ may change dramatically if some sender-tasks switch to other statuses of execution. The delay time for recipient-task i becomes longer (i.e. $\Delta P_{i,S_i}(t)$ becomes smaller), if at least one of its sender-task(s) switches to the 'accurate' status, and shorter if this switch is opposite.

3.3 Client Utility

In our model, each task is *near-continuous* [7, 8], and hence each task i has periods of interruption $\tau_{i,l}^{int} \in \mathbb{R}$ and execution $\tau_{i,l}^{exe} \in \mathbb{R}$, and the pairs of consecutive interruption and execution periods $\left(\tau_i^{int}, \tau_i^{exe}\right)_l$ have a counter $l \in \mathbb{N}$ within a total duration of task execution $\tau^{tot} \in \mathbb{R}$. Each $\tau_{i,l}^{int}$ starts at $t_{i,l-1}^{end}$ and ends at $t_{i,l}^{str}$, while each $\tau_{i,l}^{exe}$ starts at $t_{i,l}^{str}$ and ends at $t_{i,l}^{end}$. τ^{tot} starts at t_{tot}^{str}, when a client has submitted initial resource requests for all its tasks, and ends at t_{tot}^{end} for all tasks. The start and end times for τ^{tot} are the same for all tasks as they are expected to be run simultaneously. If one task is interrupted, this starts affecting negatively the lower layer tasks from the same sub-tree and all upper layer tasks from the same branch at once. We also consider a *cumulative duration of interruption* $\tau_{i,l}^{cum} = \sum_{k=1}^{l} \tau_{i,k}^{int}$ for each task i which reflects on the overall success of task execution.

In addition to single and cumulative interruptions [7, 8], our model of interdependent tasks also considers inaccurate processing of data as a factor which

negatively affects client utility. That is, the longer the task is running with inaccurate input data (not running), the more substantial becomes a negative impact on client utility. The impact of any negative factor (interruption or inaccurate data processing) is designed as the corresponding damping function $SI\left(\tau_{i,l}^{int}\right)$ for a single interruption, $CI\left(\tau_{i,l}^{cum}\right)$ for a cumulative interruption and $IP\left(\hat{\tau}_i^{del}\left(t_{dl},t\right)\right)$ for a duration of inaccurate data processing $\hat{\tau}_i^{del}\left(t_{dl},t\right)$, starting at t_{dl}. The duration $\hat{\tau}_i^{del}\left(t_{dl},t\right)$ denotes a part of delay time $\tau_i^{del}\left(t_{dl},t\right)$ which has passed till time t. Each damping function produces values from the interval $]0,1]$, where 1 denotes no impact. These functions comply with our assumption that the client's estimation of the parameter becomes gradually rather than immediately unrealistic after the task's failure, which also echoes a notion of short interruption. Only execution periods contribute positively to client utility, and the amount of such contribution is affected negatively by these damping functions:

$$\begin{aligned} SI\left(\tau_{i,l}^{int}\right) &= \frac{1}{e^{\left(\tau_{i,l}^{int}-\tau_{int[i]}^{max}\left(t_d\right)\right)/\epsilon_{int[i]}\left(t_d\right)}+1}, \\ CI\left(\tau_{i,l}^{cum}\right) &= \frac{1}{e^{\left(\tau_{i,l}^{cum}-\tau_{cum[i]}^{max}\left(t_d\right)\right)/\epsilon_{cum[i]}\left(t_d\right)}+1}, \\ IP\left(\hat{\tau}_i^{del}\left(t_{dl},t\right)\right) &= \frac{1}{e^{\left(\hat{\tau}_i^{del}\left(t_{dl},t\right)-\tau_{del[i]}^{max}\left(t_{dl},t\right)\right)/\epsilon_{del[i]}\left(t_{dl},t\right)}+1}, \end{aligned} \tag{1}$$

where $\tau_{int[i]}^{max}\left(t_d\right)$, $\tau_{cum[i]}^{max}\left(t_d\right)$ and $\tau_{del[i]}^{max}\left(t_{dl},t\right)$ are inflection points, which denote the durations of time after which client utility is noticeably affected by the corresponding factor, and $\epsilon_{int[i]}\left(t_d\right)$, $\epsilon_{cum[i]}\left(t_d\right)$ and $\epsilon_{del[i]}\left(t_{dl},t\right)$ determine the speed of decrease of the corresponding damping functions around the inflection points. As $SI\left(\tau_{i,l}^{int}\right)$ and $CI\left(\tau_{i,l}^{cum}\right)$ show the impact of interruptions on client utility, their inflection points can be calculated in proportion to the damping time $\tau_i^{dam}\left(t_d\right)$. In this work, $\tau_{int[i]}^{max}\left(t_d\right)=\tau_i^{dam}\left(t_d\right)$ as the damping time shows how fast task's interruption substantially affects client utility in terms of the unobserved changes in the estimated parameter. Considering $IP\left(\hat{\tau}_i^{del}\left(t_{dl},t\right)\right)$ shows the impact of inaccurate data processing on client utility, its inflection point is equal to the delay time $\tau_i^{del}\left(t_{dl},t\right)$. The values of $\epsilon_{int[i]}\left(t_d\right)$, $\epsilon_{cum[i]}\left(t_d\right)$ and $\epsilon_{del[i]}\left(t_{dl},t\right)$ are calculated in proportion to their respective inflection points.

In our work, the *effectiveness function* $E\left(t\right)$ [7, 8] demonstrates the success of task execution over time t. This function changes only during execution periods, and it can be reduced, multiplying by the values of damping functions. First, an estimate $Es\left(t,E\left(t_{i,l-1}^{end}\right)\right)$ is linearly increasing during an execution period:

$$Es\left(t,E\left(t_{i,l-1}^{end}\right)\right)=\frac{\left(1-E\left(t_{i,l-1}^{end}\right)\right)t+E\left(t_{i,l-1}^{end}\right)t_{tot}^{end}-t_{i,l}^{str}}{t_{tot}^{end}-t_{i,l}^{str}}. \tag{2}$$

This estimate $Es\left(\cdot\right)$ starts increasing from the value of effectiveness function $E\left(t_{i,l-1}^{end}\right)$ at the end of previous execution period $\tau_{i,l-1}^{exe}$ towards the desirable

end of execution at time t_{tot}^{end} when the value of effectiveness function is equal to 1. The full effectiveness function is presented below:

$$E(t) = \begin{cases} Es\left(t, E\left(t_{i,l-1}^{end}\right)\right) SI\left(\tau_{i,l}^{int}\right) CI\left(\tau_{i,l}^{cum}\right) IP\left(\hat{\tau}_i^{del}\left(t_{dl}, t\right)\right), \\ \qquad\qquad if\ \tau_{i,l}^{exe} \neq 0\ and\ \tau_i^{del}\left(t_{dl}, t\right) \neq 0, \\ Es\left(t, E\left(t_{i,l-1}^{end}\right)\right) SI\left(\tau_{i,l}^{int}\right) CI\left(\tau_{i,l}^{cum}\right), \\ \qquad\qquad if\ \tau_{i,l}^{exe} \neq 0\ and\ \tau_i^{del}\left(t_{dl}, t\right) = 0, \\ E\left(t_{i,l-1}^{end}\right), \qquad\qquad if\ \tau_{i,l}^{exe} = 0. \end{cases} \tag{3}$$

Note that the values of $SI\left(\tau_{i,l}^{int}\right)$ and $CI\left(\tau_{i,l}^{cum}\right)$ are constants within an execution period $\tau_{i,l}^{exe}$ (i.e. $\tau_{i,l}^{exe} \neq 0$), while the values of $IP\left(\hat{\tau}_i^{del}\left(t_{dl}, t\right)\right)$ decrease within this period. $IP\left(\hat{\tau}_i^{del}\left(t_{dl}, t\right)\right)$ affects the effectiveness of task execution only when task i is using inaccurate input data (i.e. $\tau_i^{del}\left(t_{dl}, t\right) \neq 0$) from its sender-task(s). The utility U_i for each task i is calculated as the square under the broken curve of $E(t)$, i.e.

$$U_i = 1/S_{max} \sum_{l=1}^{L_i} \int_{t_{i,l}^{str}}^{t_{i,l}^{end}} E(t)\, dt. \tag{4}$$

$S_{max} = \tau^{tot}/2$ is the largest possible square under $E(t)$ if task i has no failures till t_{tot}^{end} and L_i is the number of execution periods within $\left[t_{tot}^{str}, t_{tot}^{end}\right]$.

The total client utility U_{total} is calculated as a sum of all U_i with the respective coefficients ϖ_i which denote the task's level of relevance for the client.

$$U_{total} = \sum_{i=1}^{N} \varpi_i \times U_i, \tag{5}$$

where N denotes the total number of client tasks. The sum of ϖ_i over all client tasks is equal to 1.0, where the total sum of all ϖ_i from the same tree layer is equal for each layer. In this way, the upper layer tasks have a more substantial impact on the client's utility, according to our model.

4 Re-allocation Strategy

In this paper, a novel re-allocation strategy *SimTask* for a client is proposed which allows a client to exchange the allocated resources among its own tasks by negotiating such exchange with the GRA. The tasks which lost resources can resume their execution instead of other tasks, and the tasks which have donated their resources to other tasks are called *donor-tasks*. Note that a client is only allowed to ask the GRA to re-allocate resources among its own tasks, but it cannot ask the GRA to re-allocate resources from another client's tasks. The aim of this internal resource re-allocation is to avoid too long interruptions which might lead to substantial utility loss. As long as the length of time is only

considered as a resource allocated by the GRA, then a task cannot share this resource with the other task, but it can donate this resource to the other task. Note that the scalability of this approach can potentially be increased if tasks are clustered into relatively data-independent groups with their respective trees and client agents, which is the focus of our future research.

A problem following from resource re-allocation is not only which task to stop in order to launch the interrupted one with a smaller loss in the client utility, but also to which extent the GRA is willing to make an exchange of the allocated resources between client tasks. Hence, the GRA is assumed to allow such re-allocations, but only with a penalty due to its own resource cost, i.e. a task might be allocated a much shorter donor's remainder of execution period.

4.1 Condition to Use the Strategy

A client decides whether an interrupted task has not been running for too long and, therefore, it needs to be donated resource from another client's task. The resource re-allocation from one task to another one might not be beneficial for a client, because another task has to be interrupted instead of the current one and the re-allocated remainder of execution period can be shortened by the GRA. However, if any task is interrupted for so long that its damping time $\tau_i^{dam}(t_d)$ is passed, then the client's utility will be substantially decreased. Hence, we argue that the interrupted task has to receive resources before its damping time is exceeded. Then, the condition for resource re-allocation is:

$$\hat{\tau}_{i,l}^{int}(t) > k_i^{dam} * \tau_i^{dam}(t_d), \tag{6}$$

where $\hat{\tau}_{i,l}^{int}(t)$ is the current duration of interruption and $k_i^{dam} \in [0,1]$ determines a portion of $\tau_i^{dam}(t_d)$ which becomes critical for a client's task. That is, if the duration of interruption becomes longer than a specified part of the damping time, a client starts negotiation with the GRA in respect of resource re-allocation from a chosen donor-task to this task i.

4.2 Criteria to Choose a Donor-Task

When a client decides that the interrupted task should be donated a resource from another task, then it has to choose a donor-task whose remainder (or a part of it due to the GRA's penalty) of the execution period might be re-allocated to this interrupted task. Here, a client aims to choose a donor-task which will have the least impact on the client's utility, if it loses its resources. We distinguish two criteria to choose the best donor-task, where the first one shows the duration of time which can be allocated for the interrupted task and the second one considers the dependencies between a donor candidate and other tasks.

The Execution Period's Remainder Generally, a client prefers to allocate a longer execution period for the interrupted task, and this execution period should

preferably end at the maximum of resource availability [7, 8]. In the context of this paper, it is only important to note that our previously developed negotiation strategy, ConTask, was intended for a client to start the next interruption period in the proximity of a peak of resource availability. Hence, it is desirable for the donor's remainder of execution period to have at least one maximum of resource availability. A client also considers that the re-allocated remainder $\tau_{j,l}^{rem}(t)$ can substantially be shortened by the GRA.

Hence, a client is designed to find donor-task j with a remainder $\tau_{j,l}^{rem}(t)$ of execution period which is closer to an arithmetical average $\tau_{av}^{rem}(t)$ between the minimum acceptable $\tau_{min}^{rem}(t)$ and the maximum available $\tau_{max}^{rem}(t)$ remainders among all client tasks which possess resources at time t. The maximum available remainder $\tau_{max}^{rem}(t)$ is the longest remainder available among the client tasks. The minimum acceptable donor-task's remainder $\tau_{min}^{rem}(t)$ of execution period should ideally end around the next maximum of resource availability from the current point in time. However, if all available remainders have no peaks of resource availability, the minimum acceptable remainder $\tau_{min}^{rem}(t)$ is calculated as $\tau_{min}^{rem}(t) = k^{rem} \times \tau_{max}^{rem}(t)$, where $k^{rem} \in [0, 1]$ is a chosen coefficient.

Assume $Rem_{j,l}(t)$ is the first criterion for a client to choose the best donor-task. This criterion is a function which formally reflects the client's preference in respect of the duration of the execution period's remainder as discussed above and its values are from 0 to 1, i.e. from the worst to the best donor-task. If the execution remainder is shorter than the minimum acceptable one, this function will return a negative number, which is then algorithmically substituted by 0. Although those tasks are not excluded as possible donors, they are unlikely to be chosen. This function is presented below for the donor candidate j at time t.

$$Rem_{j,l}(t) = \frac{\left(\tau_{j,l}^{rem}(t) - \tau_{min}^{rem}(t)\right) \times \left(\tau_{max}^{rem}(t) - \tau_{j,l}^{rem}(t)\right)}{\left(\tau_{av}^{rem}(t) - \tau_{min}^{rem}(t)\right) \times \left(\tau_{max}^{rem}(t) - \tau_{av}^{rem}(t)\right)}. \tag{7}$$

The Donor-Task's Dependencies A client aims to minimise the negative impact on its utility when a donor task loses its resource. Assume that a client has a list of donor candidates and each of them has some remaining execution time. However, the data from these candidates have different levels of importance in respect of their corresponding recipient-task(s). If the recipient-task of the donor candidate is running, then a client has to estimate when this task will be stopped due to inaccurate input data, considering the corresponding donor candidate is interrupted. If the data from this donor candidate is of less importance for its corresponding recipient-task, then the delay time for this recipient-task will be longer. The longer delay time means the longer task is able to run with inaccurate data, contributing into the client utility. In the case when the recipient-task i of the donor candidate has already been 'stopped' or 'interrupted', the desirable donor candidate j for a client should still be of less importance to this recipient-task as defined by the weight $\alpha_{j,i}$.

Consequently, a client prefers more as a donor that task j at time t which has the smallest level of importance for its recipient-task. This condition means that

the most preferable donor candidate should ideally have the longest remaining delay time $\check{\tau}_i^{del}(t_{dl},t) = \tau_i^{del}(t_{dl},t) - \hat{\tau}_i^{del}(t_{dl},t)$ for its recipient-task i in the case it is chosen as a donor if its recipient-task is running, or the smallest level of importance $\alpha_{j,i}$ if its recipient-task is not running among all donor candidates. Hence, a variable $Con_j^i(t_{dl},t)$ is determined at time t for each donor candidate j, which value varies from the least 0 to most 1 preferable donor candidate (this applies to other variables below).

$$Con_j^i(t_{dl},t) = \begin{cases} \dfrac{\check{\tau}_i^{del}(t_{dl},t) - \check{\tau}_{min}^{del}(t)}{\check{\tau}_{max}^{del}(t) - \check{\tau}_{min}^{del}(t)}, & \textit{when task } i \textit{ is running}, \\ \dfrac{\alpha_{max} - \alpha_{j,i}}{\alpha_{max} - \alpha_{min}}, & \textit{when task } i \textit{ is not running}. \end{cases} \tag{8}$$

where $\check{\tau}_{max}^{del}(t)$ and $\check{\tau}_{min}^{del}(t)$ would be the longest and shortest remaining delay times at time t among all running recipient-tasks of donor candidates as if those candidates were chosen as donors, while α_{max} and α_{min} are the largest and smallest levels of importance among all client tasks (not only donor candidates).

The donor candidates from the lower layers of a tree are considered to be more preferable for a client as compared to the donor candidates from the upper layers, because interruption of an upper layer task will decrease the client utility more significantly than interruption of a lower layer task. The root task indicates the highest layer $N_{lay} - 1$, while the lowest layer of a tree is identified as a zero layer. Hence, a variable

$$Lay_j = 1 - (Layer_j/(N_{lay} - 1)), \quad Lay_j \in [0,1]. \tag{9}$$

is defined, which value varies between 0, i.e. the least, and 1, i.e. the most preferable donor candidate.

A client also considers a status of execution $Status_j(t)$ (see Section 3.1) of a donor candidate j at time t. A client does not consider tasks with the status 'interrupted' as possible donor candidates. The 'stopped' donor candidates are considered to be the most preferable for a client in terms of the least negative impact on the client utility. However, if a donor candidate has the status 'inaccurate' or 'accurate' and it is interrupted, then this will affect negatively all other dependent tasks which are running without or smaller error. That is, the statuses 'inaccurate' and 'accurate' are regarded as equally non-preferable statuses. Finally, we introduce a variable $Stat_j(t)$ for a donor candidate j as:

$$Stat_j(t) = \begin{cases} 0, \, if \;\; Status_j(t) = \text{`}interrupted', \\ \lambda, \, if \;\; Status_j(t) = \text{`}inaccurate' \vee \text{`}accurate', \\ 1, \, if \;\; Status_j(t) = \text{`}stopped', \end{cases} \tag{10}$$

where $\lambda \in]0,1[$. The second criterion for a client to choose the best donor-task is a function $Dep_j^i(t_{dl},t)$, $j \in S_i$, $j \neq i$, which determines the client's decision in terms of the values from 0 to 1, considering client preferences mentioned above.

$$Dep_j^i(t_{dl},t) = Con_j^i(t_{dl},t) \times Lay_j \times Stat_j(t). \tag{11}$$

A function $Don^i_{j,l}(t_{dl}, t)$, which produces a value from 0 to 1 for each donor candidate j, combines both client criteria, $Rem_{j,l}(t)$ and $Dep^i_j(t_{dl}, t)$, as:

$$Don^i_{j,l}(t_{dl}, t) = W_{rem} \times Rem_{j,l}(t) + W_{dep} \times Dep^i_j(t_{dl}, t), \quad (12)$$

where the weights W_{rem} and $W_{dep} \in [0, 1]$ and their sum is equal to 1. In other words, a client might prioritise the execution period's remainder of a donor task over the impact of this task's interruption on a task tree, and vice verse. A client chooses a donor candidate j for which $Don^i_{j,l}(t_{dl}, t)$ is the largest at time t.

5 Evaluation

We evaluate the SimTask re-allocation strategy in terms of the client utility, compared to the case when this strategy is not used, in various Grid environments with different weights in respect of criteria to choose a donor-task. The different environments are modelled by varying the probability of unexpected task interruption and an accuracy of the client's estimation of the resource availability maximum as this accuracy shows the level of periodic determinism in resource availability fluctuations. The probability of unexpected task interruption denotes the reliability of the Grid system in terms of resource failure and / or withdrawal.

The more accurate a client is able to identify the maximum of resource availability, the more favourable conditions are for negotiation during the tasks' expected interruptions. The different priorities (see Equation (12)) over criteria to choose the best donor-task denote whether the most suitable remaining execution period $\tau^{rem}_{j,l}(t)$ or the least relevant donor candidate in respect of other tasks' execution affects the client decision to the larger extent.

In our evaluation, a client has 40 tasks which are connected hierarchically as a four-layer tree, where each task has three sender-tasks (if applicable) respectively. The values of $\alpha_{i,j}$ are generated randomly for each test, where all tasks have to be run continuously and simultaneously for $\tau^{exec}_{dl} = 300000$ virtual seconds. The period of change in resource availability is equal to 3000 virtual seconds. The average client utility is then calculated over 200 runs. Note that k^{dam}_i (see Equation (6)) and λ (see Equation (10)) are set to 0.5 and 0.6 for all tasks.

A possibility for a task to obtain a longer duration of an execution period is simulated, following a periodicity of resource availability, where these durations fluctuate periodically over time. The probability of successful negotiation also increases when resources are more available. We also assume that in the resource re-allocation negotiation the GRA is less greedy than in the ordinary negotiation, because it re-allocates resources which are granted to a client. Hence, the re-allocation negotiation has a high probability of succeeding. The change of an estimated parameter $P_{i,S_i}(t)$ can be modelled with any functional dependence which satisfies the condition stated in Section 3.2. For transparency, it is modelled as a periodic function over time, considering that this parameter should not change abruptly (e.g. temperature).

5.1 Grid Environments

In this section, we evaluate the change in the client's utility for the different probabilities of unexpected task interruption and levels of accuracy with which a client estimates the maximum of resource availability. These different settings simulate more or less favourable Grid environments for negotiation. The probabilities of unexpected task interruption are considered in the interval between 1.E-02 and 1.E-06. The probabilities larger than 1.E-02 are considered to be non-realistic, because all tasks would be interrupted almost every virtual second. Figure 1 supports this assumption as it shows that the client utility changes insignificantly above the probability 5.E-04 and it generally tends to zero towards the larger probabilities. This occurs due to the fewer number of unexpected task interruptions which is approximately the same for such small probabilities and any possible difference averages over multiple runs.

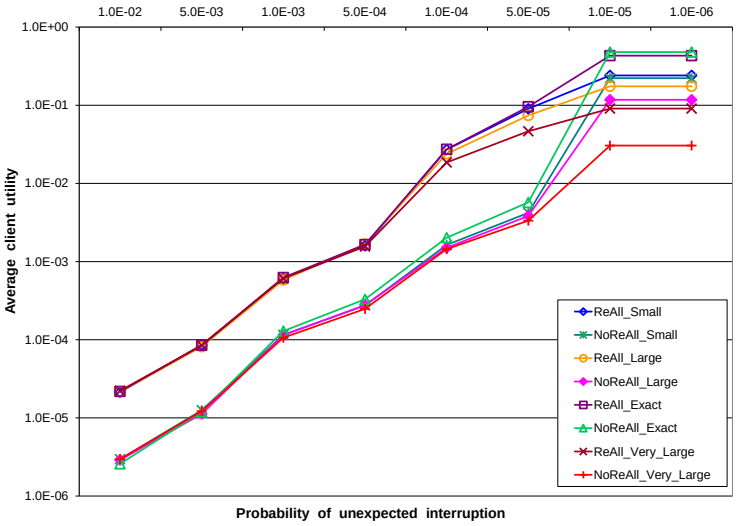

Fig. 1. The changes in the client utility in the different Grid environments

The different level of accuracy with which a client is able to estimate the maximum of resource availability mean that a task might stop running farther from the maximum of resource availability due to the client's inability to estimate this maximum accurately. Then, it will be more challenging for a client to obtain an acceptable execution period through an ordinary negotiation with the GRA. In this case, a client is more likely to use the SimTask re-allocation strategy in order to run the interrupted tasks. We consider four different levels of accuracy in the estimation of the maximum resource availability by a client, where a precise estimation is indicated as 'Exact', while an inaccurate estimation with a small deviation is indicated as 'Small', with a large deviation as 'Large', and

with a very large deviation as 'Very_Large' in Figure 1. Here, a small deviation (positive or negative) from the maximum resource availability is considered to be up to 1% of the duration of a period (one virtual day) of resource availability fluctuation. Large and very large deviations denote up to 2% and 4% of the duration of this period respectively. Finally, we compare the cases when a client uses the SimTask re-allocation strategy (i.e. 'ReAll') and when it does not use this strategy (i.e. 'NoReAll'). In the cases 'ReAll', the weights which are used to choose the best donor-task are $W_{rem} = 0.3$ and $W_{dep} = 0.7$ (see Equation (12)), and these weights is the most successful combination among all considered combinations as discussed in the following section.

Figure 1 shows the average client utilities for the different probabilities of unexpected task interruption in a logarithmic scale. Note that the SimTask re-allocation strategy improves the client utility in almost all presented cases, except for the two smallest probabilities when the maximum resource availability can be estimated precisely. An effectiveness of the re-allocation strategy decreases when the number of unexpected interruptions drastically drops and negotiation conditions are favourable in terms of resource availability. However, in the cases of small, large or very large estimation deviations, the SimTask re-allocation strategy shows a noticeable improvement (especially, for the larger deviations) in the client's utility over the cases when this strategy is not used.

5.2 Client Priorities

We evaluate how the priorities for the two criteria which are used to choose the best donor-task, might affect the client's utility for the different probabilities of unexpected task interruption. Here, we consider the different values of W_{rem} and W_{dep} for the SimTask re-allocation strategy 'ReAll', which is compared to the cases when this strategy is not used, i.e. 'NoReAll' and 'NoReAll_NoMax'. The case 'NoReAll_NoMax' also considers that a client cannot estimate the maximum resource availability, while all other cases assume that a client can estimate it with high precision. Figure 2 shows the average client utilities for the different weights over various Grid environments, where W_{rem} and W_{dep} are indicated on the labels 'ReAll' e.g., 'ReAll_0.0_1.0' denotes $W_{rem} = 0.0$ and $W_{dep} = 1.0$.

Generally, the utilities for the case $W_{rem} = 0.3$ and $W_{dep} = 0.7$ are larger than for all other combinations of the weight coefficients. Note that a strict prioritisation of one of the criteria, i.e. 'ReAll_1.0_0.0' or 'ReAll_0.0_1.0', generally show the smallest utilities among all weights' combinations. However, 'ReAll_0.0_1.0' demonstrates the larger utilities for the smaller probabilities (below 5.E-04). Hence, it is more beneficial for a client to prioritise the criterion $Dep_j^i(t_{dl}, t)$ a bit more over the criterion $Rem_{j,l}(t)$, i.e. 'ReAll_0.3_0.7'. Note that the difference in utilities is not large for the cases where the weights are more balanced such as 'ReAll_0.3_0.7', 'ReAll_0.7_0.3' and 'ReAll_0.5_0.5', while 'ReAll_0.3_0.7' usually shows the better utilities. However, the best choice of those weights might depend on a use case. Finally, the SimTask re-allocation strategy with any weights' combination outperforms almost all cases 'NoReAll_'.

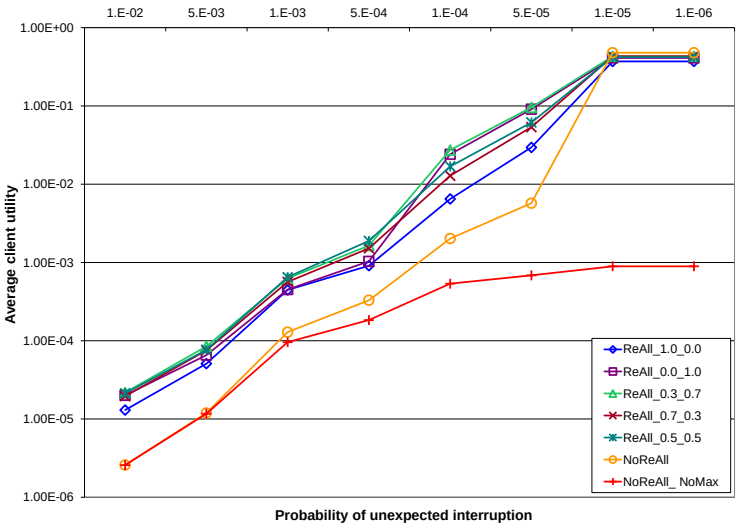

Fig. 2. The changes in the client utility with the different preferences criteria

6 Conclusions and Future Work

This paper presents a formal model for inter-dependent continuous tasks, where some tasks depend on data from other tasks. This model takes into account not only a direct data dependence between a sender and recipient-tasks, but also a reverse dependence when a sender-task is stopped due to the interruption (stopping) of its recipient-task. If a recipient-task does not receive data from some (all) of its corresponding sender-tasks for some time, it stops due to a substantial increase in its parameter estimation's error. If one task is interrupted, it affects its whole sub-tree and all corresponding recipients up to the root.

Here, a new re-allocation strategy, SimTask, has been introduced which allows a client agent to re-allocate resources among its own tasks through negotiation with the GRA, if ordinary resource negotiation becomes too long as resources are contested by other clients. This strategy includes a decision mechanism with two criteria to choose a donor-task if necessary. These criteria consider the execution period's remainder of each candidate and its importance for other tasks in a tree. As evaluated, SimTask increases the client utility for almost all probabilities of unexpected task interruption with the different estimation accuracy of the maximum resource availability.

References

1. Andrzejak, A., et al.: Characterizing and predicting resource demand by periodicity mining. Network and Systems Management 13(2), 175–196 (2005)
2. Apache: Storm - distributed and fault-tolerant realtime computation. Available from: $\langle http://storm.incubator.apache.org/\rangle$ (Visited in June 2014)

3. Babu, S., et al.: Continuous queries over data streams. SIGMOD Record 30(3), 109–120 (2001)
4. Barbieri, D.F., et al.: C-sparql: Sparql for continuous querying. In: The 18th International Conference on World Wide Web. pp. 1061–1062. ACM, NY, USA (2009)
5. Ghanem, M., et al.: Sensor grids for air pollution monitoring. In: The 3rd UK e-Science All Hands Meeting (2004)
6. Haberland, V.: Strategies for the Execution of Long-Term Continuous and Simultaneous Tasks in Grids. Ph.D. thesis, NMS, King's College London, UK (2015)
7. Haberland, V., et al.: Negotiation to execute continuous long-term tasks. In: Schaub, T., et al. (eds.) The 21st European Conference on Artificial Intelligence. Frontiers in Artificial Intelligence and Applications, vol. 263, pp. 1019–1020 (2014)
8. Haberland, V., et al.: Negotiation strategy for continuous long-term tasks in a grid environment. Autonomous Agents and Multi-Agent Systems 31(1), 130–150 (2017)
9. Iosup, A., et al.: The grid workloads archive. Future Generation Computer System 24(7), 672–686 (2008)
10. Jin, H., et al.: A run-time scheduling policy for dependent tasks in grid computing systems. In: The 6th International Conference on Parallel and Distributed Computing, Applications and Technologies. pp. 521–523 (2005)
11. Kondo, D., et al.: Characterizing and evaluating desktop grids: an empirical study. In: The 18th International Parallel and Distributed Processing Symposium (2004)
12. Le-Phuoc, D., et al.: A middleware framework for scalable management of linked streams. Web Semantics: Science, Services and Agents on the World Wide Web 16(0), 42 – 51 (2012)
13. Lee, L.T., et al.: A non-critical path earliest-finish algorithm for inter-dependent tasks in heterogeneous computing environments. In: The 11th IEEE International High Performance Computing and Communications. pp. 603–608 (2009)
14. Lesser, V., et al.: Evolution of the GPGP/TAEMS domain-independent coordination framework. Autonomous Agents and Multi-Agent Systems 9(1-2), 87–143 (2004)
15. Meriem, M., et al.: Dynamic dependent tasks assignment for grid computing. In: Hsu, C.H., et al. (eds.) Algorithms and Architectures for Parallel Processing, LNCS, vol. 6082, pp. 112–120. Springer (2010)
16. Motwani, R., et al.: Query processing, resource management, and approximation in a data stream management system. In: The 1st Biennial Conference on Innovative Data Systems Research. pp. 245–256 (2003)
17. Sandnes, F.E., et al.: Stochastic DFS for multiprocessor scheduling of cyclic taskgraphs. In: Liew, K.M., et al. (eds.) Parallel and Distributed Computing: Applications and Technologies, LNCS, vol. 3320, pp. 354–362. Springer (2005)
18. Sardinha, A., et al.: Scheduling cyclic task graphs with scc-map. In: The 3rd Workshop on Applications for Multi-Core Architectures. pp. 54–59 (2012)
19. Sequeda, J.F., et al.: Linked stream data: a position paper. In: The 2nd International Workshop on Semantic Sensor Networks. vol. 522, pp. 148–157 (2009)
20. Terry, D., et al.: Continuous queries over append-only databases. SIGMOD Rec. 21(2), 321–330 (1992)
21. Wooldridge, M., Jennings, N.R.: Intelligent agents: theory and practice. The Knowledge Engineering Review 10, 115–152 (1995)
22. Yang, T., et al.: Heuristic algorithms for scheduling iterative task computations on distributed memory machines. IEEE Transactions on Parallel and Distributed Systems 8(6), 608–622 (1997)
23. Zhao, H., et al.: A low-cost rescheduling policy for dependent tasks on grid computing systems. In: The European Across Grids Conference. pp. 21–31 (2004)